# Constructing a Support Environment for Cooperative Works over a Computer Network by Integrating Software Process Enactment Support and Communication Support

Koichiro Ochimizu
March 23, 1997
IS-RR-97-0012S

School of Information Science
Japan Advanced Institute of Science and Technology, Hokuriku
Asahidai 1–1, Tatsunokuchi
Nomi, Ishikawa, 923-12, JAPAN
ochimizu@jaist.ac.jp



ISSN 0918-7553

# Constructing a Support Environment for Cooperative Works over a Computer Network by Integrating Software Process Enactment Support and Communication Support

Koichiro OCHIMIZU
School of Information Science,
Japan Advanced Institute of Science and Technology, Hokuriku
1-1 Asahidai, Tatsunokuchi, Noumi, Ishikawa 923-12, Japan
Phone: +81-761-51-1260
Fax: +81-761-51-1149
e-mail: ochimizu@jaist.ac.jp

**Abstract**

In this paper, we will propose a model and an environment that facilitate our cooperative works over a computer network by investigating an mechanism that integrates a software process enactment support and a communication support. The goal of our work is to achieve the enhancement of pleasantness of software development environments featured by the following four properties: definiteness of a software design method; independentness of an individual's work environment ; smoothness of communications; portability of one's own computing environment. In section 1, we will show the purpose and the goal of our research, addressing several research issues to be solved.

In section 2, we will examine the following three issues to make principles and philosophies clear both for defining the model and for constructing the environment.

- The role of software architectures in a software design method.
- When and under what conditions can communication occur and go well ?
- What should be described in a software process model ?

In section 3, we will introduce the state of the progress of the JIZAI project in Ochimizu's laboratory. The aim of the JIZAI project is to construct a support environment for cooperative works over a computer network, especially for software engineering activities. In section 4, we will show a next research plan.

## Contents

# 1 The purpose of the JIZAI project

What should we do now to make software technologies more emotional and sound like buildings and musics organized on clear principles ? First of all, we introduce the Kojima's essay[1] on the architecture of Italian buildings. His considerations on architecture would suggest us several useful viewpoints in considering architecture of software environments by analogy.

- *Architecture consists of the following three things: what should be expressed by buildings ?; what are the good buildings ?; and how to make good buildings ? Buildings of the ancient Roman empire expressed the power. Power was represented by the amount of materials(i.e. size). The form of a building was simplified to make us feel sizes of physical objects as a size of a building. A building of the ancient Roman empire consists of rectangular solids and cylinders where proportion of the width, depth, height are carefully examined. The technology enabled Roman people to do it was cement.*

We can say, by analogy: What should we express by a software development environment(SDE) ?; What are the good SDE ?; How can we make good SDEs ? It is important to make ourselves clear about what we want to express for creating the SDE that is sympathized by the people today and can survive in the next era. Quickness, robustness, flexibility have been investigated as the goals in the research area of software engineering. I believe one of the important goals in the next era is an enhancement of pleasantness.

It is necessary to discuss proper features of the pleasant SDE for setting up technical goals to achieve the goal. I think there are at least four features. They are definiteness, independentness, smoothness and portability. Definiteness means that we can follow a design method in a concrete and definite manner with an evaluation of design artifacts. Independentness means that we can keep the individual's working environment independent from the others but with necessary communication. Smoothness means that every member of the team can communicate with each other smoothly. Portability means that we can quickly access the necessary information from any places where we will be, or information can follow after us to every place we will visit.

# 2 Several issues to be solved in the research area of a software design method, CSCW and a software process modeling

In this section, we will explain the reason why we think definiteness, smoothness and independentness are important factors of a pleasant SDE. Portability will be discussed in section 4.

## 2.1 The role of software architectures in a software design method

Let's go back to the Kojima's essay again.

- *The style of architecture consists of layout of spaces and detail features of a building. The less-detailed architecture appeared in the era of revolution of the style and layout of spaces was reorganized to produce new one. On the other hand, however, in the era without revolution, various kinds of detail features grew.*

The layout of spaces in designing a building corresponds to a software architecture in designing a software. I believe the development of the research on a software architecture will contribute much to make a software design method definite. We examine, here, the definition of the term, software architecture.

The term, software architecture is used in quite different ways. David Garlan of CMU discussed a critical aspect of the design for any large software system (i.e. high-level organization of computational elements and interactions between those elements ) in his paper[2]. He introduced the definition of software architecture according to the definition given by the SEI research group as follows; "The structure of the components of a program/system, their inter-relationships, and principles and guidelines governing their design and evolution over time."

He classified the research efforts into two distinct trends.

1. Developing a shared repertoire of methods, techniques, patterns, and idioms among designers for structuring complex software systems. Although these terms are rarely assigned precise definitions, they permit designers to describe complex systems using abstractions that make the overall system intelligible.

2. Exploiting specific domains to provide reusable frameworks for products families. Such exploitation is based on the idea that common aspects of a collection of related systems can be built at relatively low cost by instanciating the shared design. Familiar examples include the standard decomposition of a compiler, standardized communication protocols, fourth-generation languages, and user interface toolkits and frameworks.

He also arranged the focus of concern in software architectures as follows;

- gross organization and global control structure
- protocols for communication, synchronization, data access
- assignment of functionality to design elements
- physical distribution
- composition of design elements
- scaling and performance
- selection among design alternatives

Roughly speaking, both software design methods and software architectures have the same purpose; bridging the gap between requirements and implementations. Differences are the followings. The software design method defines steps of transformation from a class of problems into the solution. The software architecture exists between the requirement and the implementation independently to help us examine issues given by D.Garlan and to help us choose the proper one from possible classes of architectures. Software design methods and software architectures can complement each other.

We have already applied existing software design methods including SA/SD and OMT to large-scaled modeling and design problems in real world to recognize the effectiveness and problems of them[3, 4, 5]. As most of design methods, however, do not define a software architecture design phase in a concrete and definite manner, we gave the following evaluation to the existing software design methods. Existing software design methods can only play a role of connecting the problem-specific solution to the classified software parts(i.e. object libraries).

We will discuss the flexibility of software architecture next. People put a high value to the size of a software when they built software monuments such as operating systems and online real time systems. Software engineering was born to control the size and the complexity of those softwares. The principle adopted was 'divide and conquer' or 'hierarchical organization of functionalities'. A hierarchical structure is organized according to the level of abstractions. It is easy to change the way of implementation at some layer to improve it, whereas it is difficult to reorganize a hierarchical structure itself because abstraction is accompanied by optimization[6]. We can not reorganize the layout of spaces only by accumulating the change of details.

Reflecting on the situation, object-oriented technologies have been studied setting up the goal toward the classification. The principle is based on the concept of type. In object-oriented design methods, though we must take the research results on design patterns into consideration, it is difficult to organize the layout of spaces because of a bottom-up manner of construction. Moreover, in the era of network, we need the new purpose and principles to deal with the flexibility of the network-wide computing structures among computing entities.

I think we can agree that the computing in the next era will take a form mentioned below.

Computing entities represented as objects interact with each other by message passing(i.e. gross organization and global control structure). Control will be decentralized and communication protocols for message sending/receiving will be standardized. Management of persistent objects would provably be centralized to maintain consistency. Communications between application objects and persistent objects will cause long-running activities[7] because of popularization of CSCW applications. Concurrency control techniques considering the delay of a network will be developed[8].

There are a lot of researches related to these subjects now. What issues will be raised when these research subjects are solved to a certain degree ?

I think one of most important issues is scaling of architectures especially in the field of supporting cooperative works over a computer network. What are the major parameters that feature the scale in architectural design. Delay is one of the important one. Delay varies depending on the scale of network and it affects the performance of the system.

It is important to promote the study on classifying software architectures based on the scale, with providing the control objects related to gross organization and global control structure, protocols for communication, synchronization, data access to give the designers frameworks and parts for a system construction. It is also necessary to define de-

sign methods and languages which enable us assign the application objects to the physical layout of the system considering the level of distribution.

## 2.2 When and under what conditions can communication occur and go well?

Groupwares gave us useful communication tools to conquer time and space distribution[9] of our communication by developing the following technologies: enhancement of realtime synchronous interaction and a feeling of togetherness made by realtime groupwares; proper presentation of information given by using multi-media; reduction of coordination cost of face-to-face meetings by using e-mails[10]. Effectiveness of the asynchronous communication (e.g. e-mail) that guarantee the completion of a communication from/to the other without interrupting one's own work is also widely accepted.

However, if they are the only benefits got back from huge investment for networks as a social infrastructure, the benefit is too small. We should explore the new application areas where we can use networks and computers to the full as a tool of intelligence augmentation in the communication.

One of the keys to exploit the new application area exists in dealing with background information of communication explicitly in the foreground. In the software development process, for example, most of the conversations are related to some part of the large amount of documents and/or knowledges of designers/programmers. Although existing groupwares can support well the activities such as noticing, asking, explaining and so on, but as for treatment of the documents, what we can do now is to display one of them on the screen and to manipulate it together. Taking large amount of documents and complex interrelationships among contents of the documents into consideration, capabilities of existing groupwares seem to be insufficient. Groupwares should provide the capabilities that can deal with a large amount of background information explicitly in the foreground, sharing them among participants.

There is a paper that point out this problem clearly. C. Ellis suggests us the research direction of groupwares in his paper[11], by defining and discussing the three user-oriented models of groupwares based on the research results in software modelling. They are; ontological model that is a static description of objects manipulated by users; coordination model that is a structural representation of activities performed by users; user interface model that is an interface representation between a system and users, or among users.

According to the ontological model, we can define necessary static information to be stored in a repository that plays a role of background information during our conversations. No one can know everything. In most of the situations, every participant has partial knowledge on the content of the repository and some of them are shared by the others. The motivation of the communication happens when someone is aware that there exists the gap of the knowledge between/among people. And then conversation would happen to remove the gap. The content of a conversation is understood by the context that each participant has. Structural context is a representation of the dynamic state of a repository. The social context and the organizational context are representation of dynamic mental states of the participants and their social background respectively. The flow of the conversation is controlled by the the context. The procedures, rules and common sense are represented by the coordination model. User Interface Model provides us the related documents, contents of communication, state of the progress of a conversation using visual representation of them.

Let us limit the range of discussion, here, to the decisions made through conversations. For example, a software development process can be regarded as a decision making process. Everyone manages the decisions made by him/herself based on the range of responsibility of his/her task. For examples, a software designer makes a specification related to design artifacts by him/herself and passes it to a programmer. A programmer designs data structures and control structures of a program referring the specification and then writes a code. The decision ( i.e. the specification related to design artifacts) should be shared between the designer and the programmer from the time of delivery. A various kinds of conversation would happen if there is a gap in understanding the content of the delivered thing. Second example is a communication between a project manager and designers/programmers. A project manager should communicate with designers and programmers on the subjects mainly related to the decisions related to time resource management.

In those situations mentioned above, we need the following information to make conversations smooth.

- Structural representation of whole decisions
- The state of the progress to the project goal
- Accumulative structure of partial decisions for each subject, and pre-conditions necessary for each partial decision
- Uncertainty and cost of each decision

These requirements suggests that it is necessary for us to develop a repository that can help us manage decisions systematically, from coarse-grained decisions to fine-grained decisions to promote a consensus making and to support change control of decisions. We must take into account of temporal features of decisions in modeling the repository. In the most of the situations, decisions are temporarily. It means an agreement would become a disagreement and a disagreement would also become an agreement as the time passes and the situation changes.

Merging the technique of decision-management into the current state of the arts of groupwares would make our long-term discussions smooth and help us produce a feeling of togetherness

## 2.3 What should be described in a software process model?

There are two major research results on the software process modeling: software process description languages and their enacting environments; the Capability Maturity Model. The former gives us a tool to design a software development process and the latter gives us both criteria to judge the developmental capability of an organization and a guideline to improve the development process[12]. In this section, we discuss the research direction of the software process modeling by presenting the author's opinions about the necessity and effectiveness of the software process modeling.

One of the aims of software process description is to formalize software design methods. Will it be successful ? In this discussion, we should carefully distinguish between the effectiveness of a software design method itself and the effectiveness of the formal description of a software design method. As most of software design methods give us only guidelines for the software development, the software process model(i.e. the formal description of a software design method ) also has the same kinds of defects. The subject we are going to discuss here is the effectiveness of the formal description itself.

Now, let's back to the point. The question: Whether the formalization of a software design method is useful or not ?; has the same meaning as the question: Whether abstraction of procedures is useful or not ?; has. A software process model consists of both a set of activities represented by 3-tuples, (operation, input, output), and control structures organizing them. By applying the concept of type and polymorphism to (ope1, di1, do1) and (ope2, di2, do2), we can get an abstract representation form of activities as (ope, di, do). In an object-oriented approach, we further combine (ope-x, di, do) and (ope-y, di, do) to get a basic representation unit as ([ope-x, ope-y], di, do). This is, however, not peculiar to the software process modeling area because it is from the programming language area. As for the control structures, we can observe several peculiar features in software process modeling. One of them is an iteration or a backtracking. In the case of software process, we can not decide the backward-jump points in looping in advance because of nondeterministic feature. It is determined dynamically [13,14].

The dynamic process evolution is also the peculiar feature of a software process modeling[15,16]. Most of human beings' behaviours are uncertain and not reproducible. Both the order of the execution and the content of a software process would often be changed and modified every software process enactment according to the mental/physical conditions of the agent(human being). Nondeterminancy and uncertainty are the peculiar features in the software process modeling.

What can we expect as the benefits of describing these control structures ? I am afraid that we do not have a concrete perspective to the point. For example, both in water-fall model and in object-oriented development approach, the iteration always happens. The difference is that the iteration is the evil in water-fall model but the good in object-oriented approach. Does the formal description of a software process enable us evaluate the defects of a water-fall model quantitatively ? Does it give us concrete and useful guidelines for prototyping approach in object-oriented software development ? Does it helpful in designing the flexible standardized development process ? Unless we can expect such kinds of effectiveness, the formalization might be just a tautological transformation from the activities of real world into another expressions.

We need to examine the origin of the problem again. The iteration happens because human beings can not make a perfect one at a time. The larger the scale of software is, the more serious this weakness of a human being become. It is difficult to predict and reproduce behaviours of human beings because human being is not a machine. We must take these human factors into account in software process modeling.

In our approach, we put our primary concern in defining the context of one's task instead of writing the procedures they must obey. Admitting the iterations as natural ones, goal of the task(e.g. quality level of products), constraints and regulations not to be violated, inter-relationships of tasks(e.g shared documents) to the other and their priorities are placed around the individual's working environment. Quality level of products is one of attributes of an object in a object base. Constrains, regulations and relationships are the attributes of task interfaces. One of the important technical issues

to be studied is a coordination support mechanism among related people when they are violated.

Will the research effort on the CMM(Capability Maturity Model) be able to give us a useful guideline to improve development processes ? F.Cattaneo pointed out in his paper[17] that the improvement of real situations is not necessarily achieved only by applying the CMM. What is wrong ?

We examine here the coordination problem between the individual's activity and the goal of the a team as one of the problem sources among several possibilities. For examples, they are: taking over another's duty or handing over one's duty; keeping the regulation and standards both of a project and an organization; producing a product satisfying the quality standards. Most critical thing is we must do every thing just in time for delivery within a limited time resource.

There is a big gap between the goal of individuals and the goal of the project team. The former admits a self-centered type of work that means one goes on with one's work within one's capability, making oneself consent, to make a good result. The latter requires a self-sacrificial type of work that means one tries to finish one's duty on a limited time. Some one can do both but it is a rare case. We developed the theory and the system that enable us to reuse the know-how of an expert using case-based reasoning and model-based reasoning techniques[18,19]. We do not have, however, any perspective to turn it to practical use because it needs huge amount of cost. We need another process model for coordination supports that can bridge between the goal of individuals and the goal of a project team.

### 2.4 Summary of section 2

In this section, first, we showed our principles and philosophies both for defining the model and for constructing the environment to support cooperative works over a computer network, especially for software engineering activities. Our goal is to achieve "Pleasantness of the environment" featured by the following four properties.

- Definiteness: we can follow a design method in a concrete and definite manner with an evaluation of design artifacts.
- Independentness: we can keep the individual's working environment independent from the others but with necessary communication.
- Smoothness: every member of the team can communicate with each other smoothly.
- Portability: we can quickly access the necessary information from any places where we will be, or information can follow after us to every place we will visit.

Next, we reviewed related research efforts in the areas of software design methods, CSCW, and software process modeling to raise the research issues to be solved. They are:

- Scaling of software architectures over a network-wide computing structure
- Merging repository control technologies, especially for decision-management, into the current state of the arts of groupwares to make long-term discussions smooth and to help us produce a feeling of togetherness
- Developing a software process model for a coordination support that can bridge between the goal of individuals and the goal of a project team.
- Human factors such as unpredictableness, not reproducibleness and imperfectness should be considered in defining the models.

## 3 The JIZAI project

In this section, we introduce the state of the progress of the JIZAI project[20] in Ochimizu's laboratory to give one of solutions for the research issues raised in section 1 and 2.

First we examine how to integrate software process enactment supports and communication support based on a decision management. Next, we propose a reference model named CSCSD model(Computer Supported Cooperative Software Development model) to support cooperative works. Third, we introduce the state of the progress of the JIZAI prototype that has the following four subsystems.

- A CSCSD server to enhance Portability: Enhancing the portability of one's own computing environment over a new computing environment that consists of Internetworking, mobile computing environments and ubiquitous information environments by absorbing heterogeneity of data. we are developing a prototype system named HISYO.
- A distribution server to guarantee Independentness: Supporting the management of shared data using control objects such as the workspace-manager and the autonomous-mediator. We have already developed the model and a prototype named GUNBU.

- A groupware base to provide Smoothness: Supporting the management of decisions by recording the progress and the reasoning of deliberations. We have already defined a scheme of a repository that consists of three layers organized by the granularity of decisions. We also developed a prototype system named SIORI.

- Active Channel to support Exploratory Learning: One of the new kinds of applications that works on the above servers. We are developing a prototype system named TANKYU.

Fourth, we will discuss the architectural issues related to the JIZAI prototype. Lastly, we will introduce several results on the protocol analysis of software modeling activities over a computer network. The purpose of the experiment is to evaluate existing four factors of plesantness and to find new factors of plesantness by observing the behaviour of human beings. We also introduce activities of the BUNSAN project that is a field test environment of JIZAI.

## 3.1 CSCSD model

What are the major elements of our cooperative works ? How are they organized to form a cooperative work ? We can enumerate at least four elements; products, communications, processes and human beings. The purpose of consideration of this section is to examine the role of each element in a cooperative work and to define inter-relationships among them. As a result of the consideration, we propose a reference model of cooperative works named the CSCSD model.

The author has already investigated the information to be managed in a software repository for supporting software engineering activities and classified them into the 3 X 3 matrix; (communication support, process management support, product management support) x ( for individual, for a team, for a project )[21]. We re-examine it here by adding the following human factors discussed in section 2:

- Necessity of support both for making long-term discussions smooth and for helping us form a feeling of togetherness.

- Kind consideration on limits of human beings' capabilities. Their behaviours are unpredictable, imperfect and not reproducible.

- Gap on goals between individuals(i.e. going on with one's work within one's capability) and a project (i.e. finishing one's duty on a limited time).

Is there any reasonable solution satisfying these requirements ? If it exists, it gives us a foundation for defining the CSCSD model. What kinds of role can computer play in executing the solution? The answer gives us a foundation for constructing the JIZAI environment.

### 3.1.1 Integration of a process model and a communication support based on a decision management

We will consider here how to record and manage the decisions, especially for decisions made by conversations by examining the relationships between products/communication and decisions produced through a software process enactment.

**On decisions**

There are two kinds of decisions in software development activities. One of them is an artifact or a product itself made by a designer or programmer and verified by a test team. The other is a result of choice from design alternatives made through developing an artifact or a product. The former one is formed by structuring the latter ones.

The latter type of decisions, the design rationale, tend to be made by conversations because of low cost of communication. For this reason, the most of researchers take a stand that it is sufficient to record the contents of conversations themselves along the flow of conversations in order to manage design rationales[22]. I think, however, it is redundant because these conversations include a lot of paraphrases and question-answering routines to achieve a good mutual understanding.

**What should be recorded?**

It is important for every participant to be able to look at the topic of a current conversation in the proper perspective to the final goal to make communication smooth. A decision tends to be changed so often because human beings can not make a perfect one at a time. So decisions are usually momentary and there is no perfect decision. We need a recording scheme that supports us to change decisions and to analyze the ripple effect of changes.

Decision management means to manage the content mentioned above in addition to the artifacts/products and their dependency relationships. How can we record and manage both types of decisions ? This is the main subject of defining the CSCSD model.

We should refer to, here, the other factors that make communication smooth. They are a paraphrasing process and a feeling of togetherness. One's ability to achieve something usually differs from ability of another person with respect to speed

and quality when they are engaged in the same cooperative problem solving activity. It is a usual case that one who achieves a goal first of all explains a point to the others carefully and throughly considering the others' ability. We call it a paraphrasing process. The paraphrasing process also occurs when they are aware of a discrepancy caused by either difference of understanding or gap of understanding. They try to resolve the discrepancy by paraphrasing their standpoints.

It seems to be useless to record conversations of a paraphrasing process because the contents of the conversation depend on ones' experience, knowledge and physical conditions. Needless to say, it is important to support a paraphrasing process because it helps us form a feeling of togetherness and help us share the state of progress and issues to be discussed next( i.e. what are decided and what are not). We think support of paraphrasing processes is a role groupwares and/or user interface.

### On adopting Coarse-grained dependency--relationships among artifacts as a basis of integration

Modeling and designing activities transform input artifacts into output artifacts/products with referring to materials such as existing products, standardized rules and quality criteria using software tools and/or human beings' thought processes. It means that there are input-output dependency relationships among artifacts/products.

Coarse-grained dependency relationships define input-output relationships among documents such as specifications, design documents and program codes. We can define a software process by adding a temporal order to the coarse-grained dependency relationships among artifacts/products. Fine-grained dependency relationships define use/used or refer-to/referred-to relationships among items included in the documents and give us a basis to understand design rationales. We adopt the coarse-grained dependency-relationships among artifacts/products as a basis for integrating definition of software process models and for recording communications.

We do not think that the coarse-grained dependency is a necessary and sufficient thing as a basis of the integration. It is, however, one of the major basis for the integration, because it provides us constraints in designing the execution order of task and gives us outline of expected conversations.

### On defining a task and a software process based on the coarse-grained dependency-relationships

A subset of artifacts/products is assigned to one of team members according to his/her role and range of responsibility. We call this subset a task. Each task has attributes, for example, rules and forms for describing artifacts/products, constraints on performing the task, interface definitions to the others' tasks, pre/post conditions of the task, quality criteria. As the details of an attribute definition depends on organizations or projects, we consider here the general features common to every task attribute definition to be useful for a project management. We think, at least, we need information required by the following functions.

- Constraints not to be violated should be reasonable and should be expressed clearly.
- A system should be able to detect constraint violations.
- A system should be able to give us an instruction in the case of constraint violations.
- A system should be able to support coordinations among related people if a treatment would cause wrong side-effects to the other people.

We define a software process as a lattice of tasks by adding a temporal order to the coarse-grained dependency relationship among artifacts/products. In defining the software process, there is a strong difficulty in balancing time resource assigned to the project and time resource required by the individual depending on one's capability and capacity.

### On recording decisions

The flow of conversations represents a transition of participants' concerns. At beginning of the flow, a conversation starts from the the most concerned issue and goes on. In the middle of the conversation, it sometimes occurs to move to the other subject for a time because someone is aware of existing of unsatisfied pre-conditions for the original issue. At the end of the conversation, another issue to be discussed successively is raised when the original issue is solved. It also the usual case that we can not get to the conclusion during the conversation and suspend it. Talking another issue, we sometime suddenly recognized the solution, then we resume the suspended conversation again. How can we record the decisions made through the flow of conversations mentioned above ? It is easy to record conversation itself along the flow of conversations like a diary. It is, however, the worst style of recording because it can not represent a cause-effect relationship among decisions explicitly.

We often experience the situation during a conversation where we review the conversations so far by arranging what are solved or not and what to be discussed next. We usually arrange the following information at this situation.

- a whole view of decisions made or to be made
- proper perspective of the current situation in the whole view
- the state of progress for the final goal
- issues to be discussed next

It is better to record the above four items.

Now we must take it into account that the structure and the content mentioned above might change in progress of the discussion. It also often occurs we can become aware of the imperfection of some decisions later. We should record the following information that can support restructuring of discussion or resteaming of a previous discussion.

- For each subject, an accumulative structure of partial decisions, and pre-conditions required to make each partial decision
- Uncertainty of each decision and cost required for it

Moreover, it is obviously necessary to record

- a correspondence between conversations and software objects

because a subject of conversation is caused by software object.

We introduce here three-layers recording scheme:a deliberation space(the top layer), a deliberation process(the middle layer), a deliberation type (the bottom layer).

- We represent a cause-effect relationships among decisions as a deliberation space. The role of a deliberation space is to give us a whole view of decisions and to help us understand both proper perspective of the current conversation to the final goal and the state of progress to the final goal
- We manage a series of conversations for an issue by a deliberation process. The role of the deliberation process is to manage an accumulative structure of partial decisions and preconditions required to make each partial decision
- We manage a series of utterance in a conversation by a deliberation type such as transmitting and adjustment, decision and creation[23]. Most of conversations during a software development process can be represented by the following three types: transmitting some decision or knowledge from the person who know it well to the person who does not know it well; selecting a candidate for the solution from several alternatives by hearing what the other has to say each other; creating some new information by combining the partial knowledges each participant has. The role of a deliberation type is to manage attributes of a decision such as uncertainty of a decision and cost required for a decision. Each deliberation type has several states and we can judge uncertainty of a decision and/or degree of achievement by seeing at what state the conversation suspended or finished. Time attribute and numbers of utterances can represent a cost of decision.

We call a repository with the above scheme groupware base.

### On relation between an enactment of a software process and communications occurred

Does a cause-effect relationship among decisions made through conversations have the same structure as an execution order of tasks has ? A structure of a deliberation process does not always match to a structure of an execution order of tasks because some issues happen depending a state of a software development as mentioned later. Therefore we manage a cause-effect relationships among decisions and an execution order of tasks separately.

#### an example

*correspondence between a deliberation type and a task*

In a software process enactment, most of communications occur when multiple people are assigned to the same task or artifacts are shared by several tasks directly/indirectly. There are several types of communication. Followings are typical examples.

- In the case that multiple people assigned to the same task, communication happens: to exchange ideas and expertise with each other; to choose a candidate for the solution among several alternatives; to compensate the gap of skills or experiences related to the task.
- In the case of artifacts shared by several tasks, communication happens: to deliver some artifact from one to the other; to make a common understanding and a consensus related to the content of delivered thing; to detect and correct errors caused by the incompleteness of artifacts.

In these cases, an instance of deliberation type corresponds to either a task or a connection of tasks directly.

*correspondence between a deliberation space*

*and an execution order of tasks*

A functional requirement specification consists of requirements. At the design phase, each of functions is assign to either a fast processor or a slow processor according to the performance requirement specification and designer's expertise. Each of two groups of functions become a software module respectively. After deciding module interfaces, global data, data structures and code of each module, each module is implemented.

Suppose the load module can not fulfill the desired performance at the integration test or *alfa* test. We must change the following two kinds of decisions already made.

- coarse-grained level of decisions: a functional requirement specification, a performance requirement specification, selection of processors(a system specification), two modules, a load module.
- fine-grained level of decisions: assignment of a function to one of two processors, number of modules, content of interface specifications among modules( i.e. number of arguments, argument type and so on), selection of data structures and program logics.

We must re-examine the decisions as follows; Was the selection of a processor right ? Was the assignment of a function to a processor right ? Was the selection of a data structure and a logic right ?

After re-examination, we should change the related specifications and programs. It is better to evoke a new deliberation process and its sub-deliberation processes for each document to be changed to proceed change in organized manner.

As shown in the example, it is necessary to evoke a deliberation process not corresponding to a component of a software process model in the occurrence of unpredictable extraordinary situation. In general, a cause-effect relationship has a different structure as an execution order of tasks has.

#### 3.1.2 CSCSD model for cooperative works

The central issues for modeling the CSCSD model are as follows:

- Management of tasks that define individuals' responsibility and management of an execution order of tasks(software process) based on the coarse-grained dependency relationship among artifacts/products.
- Management of cause-effect relationships among decisions(groupware base) based on fine-grained dependency relationships among the items included in the documents.

We organize the above functions as a hierarchical structure as follows.

- At the lowest layer we put the CSCSD server that enable us a network transparent data access with resolving heterogeneity of data.
- At the first layer from the bottom, we manage the dependency relationships among artifacts/products. Functions of this layer support to define guidelines of works and constraints related to tasks for constructing the distribution server in the upper layer and also give a groupwares the basis for recording the contents of communication by keeping links between contents of conversations and artifacts or dependency relationships of artifacts.
- At the second layer from the bottom, we put a distribution server, a groupware base. The distribution server consists of a process server and workspaces. The process server define tasks, an execution order of them, and resources available. The workspace defines a group of related artifacts according to one's responsibility and manage shared data between tasks. The groupware base keeps a cause-effect relationship among decision made through conversations.
- At the third layer from the bottom, we put CASE tools and groupwares to support both producing artifacts and making conversations like consensus making or delegation.
- At the fourth layer from the bottom( The top layer), we put a UIMS to display information related to various kinds of servers and tools. Context information proposed by Ellis will be displayed if necessary.

### 3.2 Current states of the progress of the JIZAI prototype

In this section, we will summarize our prototyping efforts on the JIZAI project. We are developing the several prototypes based on the goal introduced in section 1 and the CSCSD model defined in section 3. What parts of functions should be executed by computers in performing the functions defined in the CSCSD model. How should we properly define the interfaces between the works done by human beings and tasks executed by computers ? We will examine first these issues for each layers of the model.

- CSCSD Server(the lowest layer): It is desirable to automate most of the functions as possible as we can. It also desirable to make the function calls transparent for users.

- Management of Dependency Relationships among Artifacts(the fourth layer): It is necessary to adopt DBMS technologies both for representing a scheme related to the order of tasks and for maintaining consistency of data access such as creating, deleting and changing.

- Distribution Server and Groupware Base(the third layer): we should contrive to use software agents(i.e. control objects) to reduce users' effort. It is also necessary to design and implement a recording scheme of a groupware base that can deal with a side-effect analysis of a decision change.

- Tool Kits (the second layer): We need a tool integration mechanism for CASE tools and groupwares to guarantee the succession of works.

- User Interface (the top layer): It is necessary to use graphics to represent structural contexts, social contexts and organizational contexts by defining and using new metaphors instead of multi-windows paradigm. We need to develop the new UIMS to support it.

#### 3.2.1 A CSCSD server to enhance Portability

An infrastructure for the new computing environment such as internetworking, mobile computing environment, ubiquitous information environment are growing now. What kinds of new possibilities will these environment bring us ? In a word, we can quickly access the necessary information from any places where we will be, or information can follow after us to every place we will visit. One of the features of these new computing environment is heterogeneity of data. The purpose of development of a CSCSD server in JIZAI project is to offer the infrastructure that can fully utilize the possibilities of internetworking and wireless LAN technologies, mobile computing environments and ubiquitous computing environments.

There are several points to be considered in constructing the server. They are, for examples, treatment of heterogeneity of data and asynchronous operations. Gio Wiederhold proposed the Mediatted Architecture that is an extension of a traditional client-server model by putting a mediator between clients and a server[24]. Mediator supports : accessing to proper information resources; data selection and format conversion; arranging data in common abstract level of representation; integration of informations from different information resources; preparing meta information according to purposes of applications. These are called middlewares and we have already a lot of products such as CORBA(Common Object Request Broker Architecture), DOE( Distributed Objects Everywhere), ILE(Inter-Language Unification), KQML(Knowledge Query and Manipulation Language), OLE(Object Linking and Embedding), OpenDoc(Open Document Exchange), PDES( Product Data Exchange using STEP). Standrarization efforts are now going on(see http://isx.com/pub/I3).

S. Heiler address interoperability problems in large-scaled distributed system[25]. He pointed out that it is necessary to guarantee semantic interoperability to exchange data and services in a heterogeneous system configuration in addition to agreement on message passing protocols, procedure name, error code, types of arguments. He showed an interesting example that the probability that two database designers, even domain experts, will choose the same element names for the same data is only between seven and eighteen percent.

Frank Manola also discussed interoperability issues in large-scale distributed object systems[26]. Classical and traditional construction methods of information systems have been matured(e.g Datarun method by D.Pascott[27]). They are powerful and useful in constructing a system for a homogeneous world. However, in large-scale distributed system, it is necessary to guarantee exchange of heterogeneous data. Interoperability of data representation and object interfaces should be certified in every level of services such as database, common services, application.

we summarize functional requirements of the CSCSD server as follows. It should be able to treat with:

(1) **scaling:** easy to select proper global control structures, communication protocols, ways of synchronization and data access according to the scale.

(2) **heterogeneity:** guarantee of data selection and format conversion in accessing proper information resources.

(3) **multi-computing environments:** executable on multi-platforms.

(4) **flexibility of architecture:** designed by an object-oriented architecture

#### 3.2.2 A Distribution Server to guarantee Independentness

We are studying management of shared information as one of the necessary conditions to achieve the goal 'independentness'. By the term "independentness", we mean one can do one's work without

unnecessary communications and interaction to the others. We have already succeeded in developing several control objects named a workspace-manager and an autonomous-mediator which can support us to manage shared data in distributed software development[28,29]. The major features of them are as follows;

1. A workspace-manager object and an artifact object which can support us(i.e. software engineers ) to manage their range of responsibility and control of data sharing.
2. An autonomous-mediator object can support negotiation among software engineers occurred through the modification of shared data.
3. Each object has a meta-object that supports us to select an available proper action dynamically according to the situation.

Using these objects, a software engineer will proceed his work, only using both the directly related knowledge to his own responsibility and the direct relationships to the others who share data with him. Our environment will support us to change policies related to data sharing in a cooperative and flexible manner.

Next research step is to add some degree of intelligences to the workspace- manager and the autonomous-mediator by applying the results on the Trio Environment[30,31]. The research results of the Trio Environment would be extensible to the software process modeling by considering the concurrency control and communication issues among objects. For example, in Trio Environment, behaviours of objects are formulated by capturing the following semantics; "As a consequence of event A, event B must occur within X seconds."

There are two points to be discussed when we want to apply the results to software modeling. First, WFFs are expressed with respect to single current time in the Trio model. In the case of software process modeling, it is natural to define that each object has own current time (i.e. starting time of job) with preconditions. It is also necessary to define their communication during software process enactment. It is the concurrency control and communication control issues. Second, in the case of software process modeling, "must" and "within" are just a expectation of a project manager, because behaviours of human being are not predictable and can not be reproduced even if agents are the same.

Therefore, we need to formulate the following semantics regarding to conflicts resolution; "If one of the objects violates the assigned constraints, the other objects should be able to detect it, and if possible, they must find the solution by negotiating with each other with help of human beings."

#### 3.2.3 A Groupware Base to provide Smoothness

We are also developing a theory and a system for decision management[32] to make the communication among team members smooth. We have already proposed a framework and schematic representations for recording the progress and reasoning of deliberations. The aim of our scheme is to help us grasp the whole view of decisions easily and to permit us control the change of decisions efficiently, by managing the decisions produced over a long-term discussions. Our framework is organized into a hierarchy consisting of three layers: a deliberation space; a deliberation process; a deliberation type [23].

- A deliberation space records the cause-effect relationships among subjects to represent the whole view of decisions and the degree of achievement for a goal of a group.
- A deliberation process represents the state of progress of a subject by recording a series of partial decisions and their preconditions.
- A deliberation type manages the related utterances in a conversation by grouping them by using recording types of a conversation such as Transmission and Adjustment, Decision, and Creation. It can also record degree of firmness and cost of a partial decision.

Our scheme supports us to manage decisions made and changed through conversations by organizing decisions systematically from a global view to a detailed level. We call the database managed by this scheme the Groupware Base.

In the design of the groupware base, we were especially careful for the following point. When we make decisions of various degrees of granularity, agreement would be momentary very often. It is usual that an agreement changes to a disagreement or a disagreement changes to an agreement when a situation change.

We have already designed and implemented a prototype system "Siori"[33], for electronic meetings using mailing lists, based on the groupwarebase model. Its purpose is to facilitate the coordination among participants and to reduce the possibility of occurrence of redundant conversations caused from an insufficient management of decisions.

#### 3.2.4 An Active Channel to support Exploratory Learning

The advanced software development environment using JIZAI also supports our daily activities over a computer network to make good softwares[34].

They are: Information transmission and acquisition by news and e-mail; World-wide discussions with related researchers; Consensus making, decision making, Cooperative works via tele-conferencings; Realtime presentation of research results. An application program to satisfy this purpose is an active channel for ubiquitous information.

Technical innovation of information retrieval technologies will occur in near future based on information filtering techniques that support us to find and use only necessary information from huge amount of data scattered over a computer network. Various kinds of information is now being accumulated on world-wide spread nodes of networks due to popularization of WWW. We call these information ubiquitous information. It will be one of important applications that supports exploratory learning, where a user with limited knowledge on information acquisition can find necessary information from ubiquitous information scattered over a computer network with help of a computer. We call the application(i.e. a tool that support exploratory learning) an active channel. A channel means a link for information filtering generated dynamically for each information retrieval request. Term active means that increase in information useful for information retrieval.

Basic functional requirements for exploratory learning is proposed and discussed by Denning[35]. Denning classified the task domain of CSCW into three domains; method, media of information exchange, exploratory learning and discussed the role of software agents in each domain.

**method:** we use CSCW as a method to get a known output in the known domain.

**media:** we use CSCW as media when we do not know an output in advance, but know the method to get it. Goal is achieved by the interaction of users.

**exploratory learning:** user does have a limited knowledge both for task domain and issues and does not have a specific output

We are now trying to design and implement the function "activeness".

## 3.3 Architectural design of the JIZAI prototype

We consider here a gross organization and global control structures of the JIZAI prototype with discussing how to assign the functions of the CSCSD model defined in section 3.1.2 to a computing environment. As we have not yet done a consideration on scaling of architectures, we suppose here the internet environment in JAPAN.

### 3.3.1 Displacement of servers and databases

- **Displacement of CSCSD servers:** There are three possible plans for displacements of CSCSD servers. They are:

  1. put each CSCSD server to each database;
  2. provide independent CSCSD servers to treat with information
  3. retrieval requests over several databases;
  4. put each CSCSD server for each user.

  We will examine the merits and demerits of these three ways of displacements. The merit of displacement(1) is that we can customize each CSCSD server for each database management system to enhance the efficiency of data access to each database. The overhead of communication among servers, however, increase and the system performance would be reduced because information retrieval requests would be done over several databases. It is also necessary to do consistency checking over a network.

  Next we will examine the displacement(2). Displacement of only one CSCSD server is not feasible because of the workload of the CSCSD server. We must decide how many CSCSD servers are necessary under what criteria. We think it is reasonable to put a CSCSD server to each project taking account into the long transaction natures of software engineering activities(i.e. the same data tends to be referred/updated for long duration of time). We should provide cash memory with each CSCSD server to guarantee the efficiency. Consistency checking should be treated by each CSCSD server.

  There is only one merit of displacement(3). Data access over a network would be reduced because data necessary for some user would be put in the cash memory of the CSCSD server.

- **Displacement of artifacts and their dependency relationships:** It is proper to put related artifacts and their dependency for each project. We must decide, however, which is better providing an independent server for them or putting them in the workarea of a CSCSD server. We think it is better to put them in an independent server physically closed to a project manager because dependency relationship among artifacts is rather static and usually updated by the project manager.

- **Displacement of distribution servers and groupware bases:** A distribution server is logically placed for each user according to his/her role. It is better to place it physically closed to the user to guarantee the quick response. As for a groupware base, it better to distinguish the following two cases: to help us understand the representation of a whole of decisions; and to help us perform heavy discussions in software modeling and software reviews. In the former case, it is enough for us to refer the content of the groupware base by using a WWW server. In the latter case, we think it is better to place SIORI server for each deliberation process to improve the efficiency.

- **Displacement of tool kits:** This is obviously put in the individuals' environment.

- **User interface issues:** UIMS is obviously put in the individuals' environment. We must be careful for placing the information related to the context proposed by Ellis because displaying those information consumes the lot of computing powers. As for structural context, we must decide the place related to the placement of a CSCSD server and a distribution server. Placement of information related to an organizational context should be decided related to the placement of groupware bases. All of the information related to contexts have a short lifetime and requires high degree of realtime responseness. Detail consideration is left to the future works.

#### 3.3.2 On an object-oriented architecture of the JIZAI prototype

We are going to construct the JIZAI prototype using an object-oriented architecture to increase the flexibility of the architecture. On this issues, we have only developed the workspace-manager and autonomous-mediator as constituents of the JIZAI architecture. Most of the works to be done are left in future.

### 3.4 Understanding human beings' behaviour by protocol analysis

In this paper, we have discussed the enhancement of pleasantness of an environment featured by the four properties: definiteness of software design methods; independentness of individual's working environment; smoothness of communication among participants; portability of one's own computing environment. We should have methods and metrics to evaluate the goodness of models and prototypes. We think protocol analysis is one of the useful methods to do it. There are two purposes of our protocol analysis experiments.

- finding another useful metrics for evaluating pleasantness

- preparing the evaluation methods for independentness and smoothness

The former means we are going to re-examine the features of plesantness including existing four features such as definiteness, independentness, smoothness and portability. The latter means we are going to develop how to evaluate them. In the JIZAI project, protocol analysis experiments , mainly related to the evaluation of smoothness of communication, have been performed to understand the problems of existing groupwares quantitatively[36].

### 3.5 The BUNSAN project as a field test environment for JIZAI

BUNSAN project is a cooperative work among JAIST, NAIST and TITECH from academies and PFU from industries[37]. The purpose of BUNSAN project is to provide a network-wide field test environment to examine fundamental research results in laboratories. The role and aim of JAIST is to have a field test environment for JIZAI. The following experiments have been performed in the past experiments of the BUNSAN project.

- Remote presentation, Remote cooperative formal specification review, Remote cooperative programming activities: using existing multicast tools for communication and the network environments such as Internet and ATM. The purpose of the experiments was to discover the problems of existing multicast tools and network facilities. We have already finished a series of first-term experiments, and have got several useful knowledges about the delay problem of networks, the communication protocol problems among human beings, that will be useful to develop new technologies related to cooperative works over a computer network.

- We performed several protocol analysis experiments to find obstructions in cooperative work over a computer network.

The next goal of JAIST in BUNSAN project is the improvement of prototype systems of JAIZAI such as GUNBU, SIORI and HISYO.

### 3.6 Summary of section 3

In this section, we proposed the CSCSD model as a reference model of cooperative works consisting of the following functional layers.

**User Interface:** this layer supports us to form a feeling of togetherness among participants of cooperative works by showing the structural context, social context and organizational context.

**Tool Kits:** this layer contains CASE tools that can automate our transformation processes and groupwares that support us paraphrasing process. It also contains new application tools such as active channel. A tool integration mechanism guarantee the succession of works using these tools.

**Distribution Server and Groupware Base:** a distribution server support us to manage workspaces and to change shared data for the purpose of increasing the independentness of one's work. A groupware base support us to manage the decisions made through a software development process

**Management of Dependency Relationships among Artifacts:** this layer supports us to manage dependency relationships among artifacts/products. It also support us to manage software process models(i.e. scheme related to the execution order of tasks) and to maintain consistency when its instances are created/deleted/changed.

**CSCSD Server:** It supports us to deal with scaling of networks, heterogeneity of data, treatment of multi-platforms.

We also introduced the several on-going prototypes. Finally we made a basic consideration on the architecture of JIZAI prototype.

## 4 Summary and Future research issues

In this paper, first, we showed our principles and philosophies both for defining the model and for constructing the environment to support cooperative works over a computer network, especially for software engineering activities. Our goal is to achieve pleasantness of the environment featured by definiteness, independentness, smoothness, and portability. The research issues to be solved for achieving our goal are as follows:

- Scaling of software architectures over the network-wide computing structures
- Merging repository control technologies, especially for decision-management, into the current state of the arts of groupwares
- Developing a software process model for coordination supports
- Consideration on human factors such as unpredictableness, not reproducibleness and imperfectness

We proposed the CSCSD model for cooperative works based on the decisions management. The CSCSD model consists of six layers hierarchical structure. They are user interface , tool kits, distribution server and groupware base, management of dependency relationships among artifacts, CSCSD server.

We have developed the distribution server and the group ware base of a JIZAI prototype now. Future works to be done are the followings to realize the new software development style and its environment.

- Further research survey on software architecture, especially on scaling problems.
- Research survey on flexible software architectures and object-oriented software architectures.
- Further deep considerations on the following issues based on the above two surveys;
- definition of classes of a software architecture
- arranging parts of a software architecture(i.e. control objects)
- studies on gross organization and global control structure
- Case studies of object-oriented software development to make software architecture design phase clear and concrete.
- Completion of development of the JIZAI prototype version 1 and evaluation of the net effect of GUNBU and SIORI.
- Research survey on software process especially on representation of policy and constraints.
- Investigation and Case studies related to decision management in real world, especially for granularity, protection and security of decisions to recognize what to be managed and how to do it.

- Further deep understanding on integration of process and communication supports to improve the CSCSD model based on the research results on the above three issues
- Establishment of the JIZAI Consortium with researchers both from industries and academies to promote technology transfer and to produce a new useful software environment as one of the major activities of the Software Colab.

**Acknowledgements**

I would like to express my best thanks to professor Calro Ghezzi and professor Stefano Ceri for their useful comments and discussions to this report.

**references**

1. T.Kojima:"Architectures", Italia, Shinchou-Sya (in Japanese).
2. David Garlan, "Research Directions in Software Architecture", ACM Computing Surveys, Vol.27, No.2, pp.257-261,June 1995.
3. Y.Kuraya, C.Kadowaki, Y.Nishiyama, K.Ochimizu:" A Case Study of Structured Analysis - Production Control System", Seamail Vol.8 NO. 8-9, pp.3-40, 1994 (in Japanese).
4. Y.Kuraya, M.Higashida, K.Fujieda, J.Suzuki, K.Ochimizu: "A Case Study of Object-oriented Analysis - Production Control System", Seamail Vol.8 NO.8-9, pp.41-76, 1994 (in Japanese).
5. K.Ochimizu, M.Higashida:"Object Modeling", Just System, 1995 (in Japanese).
6. K.Ochimizu:"Fundamentals in Practicng Software engineering Principles –Analysis, Design and Programming", Nikka-Giren Syuppan, 1994 (in Japanese).
7. U. Dayal, M. Hsu and R. Ladin: "A Transaction Model for Long-Running Activities", Proc. of 17th International Conference of VLDB, pp.113-122,1991.
8. C.A. Ellis and S.J.Gibbs: "Concurrency Control in Groupware Systems", Proc. of the ACM SIGMOD'89, pp.399-407, May 1989.
9. C.A. Ellis and S.J.Gibbs and G.L.Rein: "Groupware: Some Issues and Experiences", Comm. of the ACM, Vol.34, NO.1, pp.38-58, Jan. 1991.
10. Martha S Feldman: "Constraints on Communication and Electronic Mail", Proc. of CSCW 86, pp.73-90, 1986.
11. Clarence Ellis, Jacques Wainer: "A Conceptual Model of Groupware", Proc. of CSCW 94, pp.79-88, 1994.
12. K.Ochimizu:"Survey of Research Activities on Software Process", Journal of IPSJ, Vol.36, NO.5, pp.379-391, 1995(in Japanese).
13. A.Ohki, K.Ochimizu: "Process Programming with Prolog", Proc. of 4th ISPW, pp.118-121, 1988.
14. K.Kishida, T.Katayama, M.Matsuo, I.Miyamoto, K.Ochimizu, N.Saito, J.H.Sayler, K.Torii, L.G. Williams: "SDA: A Novel Approach to Software Environment Design and Construction", Proc. of 10th ICSE, pp.69-78, 1988.
15. G.Cugola, E.Di Nitto Frank, C.Ghezzi: "How to deal With Deviations During Process Model Enactment", Proc. of ICSE 17, pp.265-273, April 1995.
16. G.Cugola, E.Di Nitto Frank, A.Fuggetta, C.Ghezzi: "A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems", (submitted to TOSEM).
17. F.Cattaneo,A.Fuggetta,L.Lavazza: "An experience in process assessment", Proc. of 17th ICSE, pp.115-121, 1995.
18. K.Ochimizu, T.Yamaguchi: "A Process Oriented Architecture with Knowledge Acquisition and Refinement Mechanisms on Software Process", Proc. of 6th ISPW, pp.145-147, 1991.
19. T.Yamaguchi, M.Kurematsu, N.Shimozu, H.Nakao, K.Ochimizu : "A Framework for Knowledge Acquisition Using Case-Based Reasoning and Model Inference(2) - Software Process Knowledge Acquisition",Journal of JSAI, Vol.11 No.4, pp.97-103, 1996.
20. K.Ochimizu, C.Kadowaki, K.Fujieda, M.Hori:"Design of A Software Development Environment JIZAI for Distributed Software Development", Technical Report of IEICE SS94-18, 1994 (in Japanese).
21. K.Ochimizu:"Software Repository", Journal of IPSJ,Vol.35, NO.2, pp.140-149, 1994 (in Japanese).

22. J.Conklin and M.Begeman:"gIBIS: A Hypertext Tool for Exploratory Policy Discussin", CSCW'88, pp.140-152, 1988.

23. C.Kadowaki, K.Ochimizu: "Recording the Progress and the Reasoning of Deliberations",(submitted to IPSJ).

24. Gio Wiederhold: "Mediation in Information Systems", ACM Computing Surveys, Vol.27, No.2, pp.265-267,June 1995.

25. Sandra Heiler: "Semantic Interoperability", ACM Computing Surveys, Vol.27, No.2, pp.271-273,June 1995.

26. Frank Manola: "Interoperability issues in Large-Scale Distributed Object Systems", ACM Computing Surveys, Vol.27, No.2, pp.268-270,June 1995.

27. D.Pascot, translated by K.Ochimizu:"Introduction of Database Design for Clients/Server Model", Nikkei BP, 1997 (in Japanese).

28. M. Hori, K.Ochimizu: "Shared Data Management Mechanism for Software Development Based on a Reflective Object-Oriented Model", Computer Software,Vol.13, NO.1, pp.37-54, 1996 (in Japanese).

29. M.Hori, Y.Shinoda, K,Ochimizu: "Shared Data Management Mechanism for Distributed Software Development Based on a Reflective Object-Oriented Model", LNCS 1080, Advanced Information Systems Engineering, pp.362-382, 1996.

30. D.Mandrioli, A.Morzenti, P.San Pietro, E.Crivelli: "Specification and verification of real-time systems in a logic framework: the TRIO environment", , Vol., No., pp.-,.

31. S.Bandinelli, A.Fuggetta, C.Ghezzi: "Software Process Model Evolution in the SPADE Environment", IEEE Trans. on SE, Vol.19 NO.12, pp.1128-1144, Dec. 1993.

32. C.Kadowaki, K.Ochimizu:"Systematic Support of Communication in Enacting a Software Process",Jaist Research Report, IS-RR-97-0009S, 1997 (in Japanese).

33. S.Konno, C.Kadowaki, K.Ochimizu:"Supporting Situation undestanding and Documentation of Communication in Electronic Meeting by Groupwarebase, Siori", IPSJ SIG on SE, 107-12, 1996 (in Japanese).

34. N.Kawakami, K.Kawase, Y.Kodera, D.Suzuki, T.Hagiwara, M.Hashimoto, K.Ochimizu: "Support Environment for Cooperative Works over a Computer Network", Jaist Research Report, IS-RR-96-0012S, 1996 (in Japanese).

35. D.Jennings:"On the Definition and Desirability of Autonomous User Agents in CSCW", CSCW and Artificial Intelligence, Springer Verlag, 1994.

36. H.Murakoshi, H.Kaiya, K.Ochimizu: "An Analysis of Obstruction in Cooperative Work over a Computer Network", Proc. of CICS'96,pp.265-267,June 1995.

37. K.Araki, K.Okamura, M.Saeki, N.Miura, K.Ochimizu, Y.Shinoda, H.Kaiya: "Towards Realization of Cooperative Works over a Computer Network", Information Processing Society of Japan, Proc. of Summer Workshop in Tateyama, pp.105-112, 1995 (in Japanese).

# ソフトウェア・プロセス実行支援とコミュニケーション支援の統合による ネットワークを介した共同作業支援環境の構築

落水 浩一郎
北陸先端科学技術大学院大学,
情報科学研究科
〒923-12 石川県能美郡辰口町旭台 1-1
Phone: +81-761-51-1260
Fax: +81-761-51-1149
e-mail: ochimizu@jaist.ac.jp

**概要**

本論文では、ソフトウェア・プロセス実行の支援とコミュニケーションの支援を統合した機構を検討することにより、ネットワークを介した共同作業を容易にするモデルと環境を提案する。我々の研究の目標は、開発方法論の明解性、個人の作業環境の独立性、コミュニケーションの円滑性、計算機環境の携帯性、の四つの特性で特徴づけられる「環境の快適性の向上」である。1章では、解決されるべきいくつかの問題点を指摘しつつ我々の研究の目的と目標を明らかにする。2章では、モデルを定義し、環境を構築するための原則と考え方を明確にするために、以下の三つの問題点を検討する。

- ソフトウェア設計方法論におけるソフトウェア・アーキテクチャの役割
- コミュニケーションはいつ、どのような条件下で成立し、かつ進行するのか？
- ソフトウェア・プロセス・モデルでは何を記述すべきか？

3章では、落水研究室で進めている「自在」プロジェクトの進捗状況を紹介する。自在プロジェクトのねらいは、ネットワークを介した共同作業に対する計算機支援環境を、特にソフトウェア・エンジニアリング諸活動を対象にして構築することにある。第4章で今後の研究課題をまとめる。

## 目 次

# 1 「自在」プロジェクトの目的

ソフトウェア諸技術を、明解な構成原理を持つ建築や音楽のように、より健全で我々の感動を呼ぶものにするためには、我々は今、何をするべきであろうか？ まず、小島氏によるイタリアの建築に関する随筆［1］を紹介しよう。彼の建築論に関する考察は、ソフトウェア環境論の考察にあたって、類推により得られるいくつかの有用な視点を我々に示唆してくれるだろう。

- 建築論とは、建築によって何を表現するか?、よい建築とは何か？、そしていかによい建築を作るか？ である。古代ローマ帝国における建築が表現し続けていたのは「力」であった。「力」は物量すなわち「大きさ」によって表現された。物理的な大きさを建築の「大きさ」として感じさせるため、形体は単純化された。古代ローマの建築は幅、奥行、高さなどの比率が注意深く検討された直方体と円筒を基本にしている。それを可能した技術はセメントであった。

類推により、我々はソフトウェア開発環境によって何を表現するか?、よいソフトウェア開発環境とは何か?、そしていかにしてよいソフトウェア開発環境を作ることができるか？ という問いかけを設定することができる。何を表現すべきかを明らかにすることは、時代の共感を呼び次の時代に生き残るソフトウェア開発環境を創出するために必要である。ソフトウェア工学の世界では、速さ、堅牢さ、柔軟性などの目標が検討され続けてきた。著者は、次の時代に重要となる一つの目標は「快適性の向上」であると考える。

目標達成のための技術課題を設定するためには、快適なソフトウェア開発環境が持つべき特性を論じることが必要である。。著者は、少なくとも明解性、独立性、円滑性、携帯性の四つの要因が必要であると考えている。明解性とは、「設計対象が評価可能であり、開発手順が明確かつ具体的であること」。独立性とは、「必要なコミュニケーションを保証しつつも、個人の作業の独立性が保持できること」。円滑性とは、「チームの構成員間のコミュニケーションが円滑であること」。携帯性とは、「活動の場から、必要な情報に迅速にアクセスできること、さらには活動の場にそれを支える情報が追随できること」をそれぞれ意味する。

# 2 ソフトウェア設計方法論、CSCW、ソフトウェア・プロセスの研究分野において解決されるべきいくつかの課題

本章では明解性、円滑性、独立性が快適性の重要な要因であると考えるにいたった理由を述べる。携帯性については第4章で述べる。

## 2.1 ソフトウェア設計方法論におけるソフトウェア・アーキテクチャの役割

小島氏の随筆に戻ろう。

- 建築の様式とは、「空間の組立て方」と「特徴的な細部」とからなる。細部の少ない建築は様式の変革期に現われ、空間の組み立て方を中心とした新しい様式が生みだされた。一方、変革のない時代には、多様な細部が発展した。

建築における「空間の構成法」は、ソフトウェア設計においては「ソフトウェア・アーキテクチャ」に対応する。著者はソフトウェア・アーキテクチャに関する研究を充実させることが、開発対象および設計方法論の明確化に最も貢献すると考えている。ここで、ソフトウェア・アーキテクチャという用語を吟味してみよう。

ソフトウェア・アーキテクチャという用語の定義は人により様々である。 CMU の David Garlan は、彼の論文中で［2］、大規模システムの設計において

クリティカルな問題、計算要素とそれら要素間の相互作用を高レベルの抽象度で構成すること、を論じた。彼は、ＳＥＩの研究グループによる定義に基づき、ソフトウェア・アーキテクチャの定義を以下のように与えている。「プログラムやシステムの構成要素の構造、構成要素間の相互関係、設計やその変更を支配する原理やガイドライン」。

彼は、ソフトウェア・アーキテクチャに関する研究動向を以下の二つに分類した。

1. 複雑なソフトウェア・システムを構造化するための手段、技法、パターン、いいまわし (idioms) などの、設計者間で共有される様々な共有レパートリーを開発する。これらの用語はシステムの厳密な定義を与えることはほとんどできないが、システムの全体像を理解可能にする抽象を用いて、複雑なシステムを設計者たちが記述するのを可能にする。

2. 一群の製品に対して再利用可能なフレームワークを提供する特別なドメインを開拓する。その根拠は、「共有されるデザインを実体化することで、互いに関連する一群のシステムの共通部分を、比較的低コストで構築できる」という考え方にある。代表的な例としては、コンパイラの標準的分解、標準化された通信規約、第 4 世代言語における共通パターンなどがあげられる。

彼はまた、ソフトウェア・アーキテクチャに関する研究の関心を以下のように整理した。

- 全体的構造と大域的制御構造
- 通信プロトコル、同期、データ・アクセス
- 設計要素への機能の割り当て
- 物理的分散
- 設計要素の構成法
- 規模への対応と性能
- 設計代替案の選択

おおざっぱな言い方をすれば、設計方法論およびソフトウェア・アーキテクチャ共、その目的は、要求と実現の間のギャップを埋めることにある。違いは以下の点にある。ソフトウェア設計方法論はある問題クラスをその解に変換するステップを定義する。ソフトウェア・アーキテクチャは要求と実現の間に独立して存在し、D.Garlan によって与えられた論点を検討し、可能なアーキテクチャのクラスから最適なものを選択することを可能にする。設計方法論とソフトウェア・アーキテクチャは互いに補完的な役割を果す。

筆者等は、設計方法論の効果と問題点を認識するために、SA/SD や OMT などの既存のソフトウェア設計方法論を実世界における大規模モデリング問題や設計問題に適用した経験を持つ［３，４，５］。しかし、ほとんどの既存の設計方法論は、ソフトウェア・アーキテクチャ設計フェーズを明解かつ具体的に定義していないので、以下のような評価を既存の方法論に与えた。「既存の設計方法論は問題依存の解を類形化された部品 (オブジェクト・ライブラリ) に接続する役割を果たすにすぎない」。

次に、ソフトウェア・アーキテクチャの柔軟性の問題を論じよう。オペレーティング・システムやオンライン・リアルタイム・システムなどの様々なモニュメントが構築された時代における価値感は、「大きさ」にあった。ソフトウェア工学は「大きさ」と「複雑さ」を制御するために生まれた。そのための原理としては「分割統治」や「機能の階層構成」が採用された。階層は「抽象のレベル分け」によって達成される。階層構成においては、ある特定の層の実装法を効率改善のために変更することは容易である。しかし、ソフトウェアにおける抽象は「最適化」を伴なうため、「階層構造自体の再構成」は困難である［６］。細部の手直しの積み重ねによっては「空間の再構成」はなしがたい。

その反省を受け、「類形化」を目標とするオブジェクト指向技術が試みられている。その構築原理は「型」の概念に基づく。デザインパターンに関する研究の成果を待った上で結論を下す必要があるが、オブジェクト指向方法論においてはシステムの構築がボトムアップになり空間の構成が困難である。さらに、ネットワーク時代においては、計算要素間の相互通信からなるネットワークにまたがる計算構造の柔軟性を保証する、新しい目標と構成原理が必要である。

次の時代における「計算」が以下に述べる形態をとることに関しては、我々は合意できると思う。

- 計算要素はオブジェクトにより表現され、その間の相互作用はメッセージ通信によってなされる (全体的構造と大域的制御構造)。制御は分散制御になり、制御およびデータの授受に関する通信プロトコルは標準化される。永続オブジェ

クトの管理は一貫性保持の目的から多分集中管理されるだろう。CSCW アプリケーションの発展により、アプリケーションオブジェクトと永続オブジェクト間の交信はロングトランザクションが主流となるだろう［７］。ネットワークの遅延を考慮に入れた書き込み制御技術も発展するだろう［８］。

上記の諸機能を実現するための基礎 /応用研究はすでに、世界の各地で実施されている。これらの研究があるレベルにまで達した時、次に問題になるのは何であろうか？

とくに、ネットワークを介した共同作業を支援する分野に話題を絞る時、重要な問題の一つとして「規模への対応 (アーキテクチャを規模対応に整備すること)」が挙げられる。アーキテクチャの設計において、「規模」を特徴づける主要なパラメータは何であろうか？ 「遅延時間」は重要なパラメータの一つである。「遅延時間」はネットワークの規模により大きく異なり、システムの性能に大きな影響を与える。

「規模」に応じたソフトウェア・アーキテクチャのクラス分けをまず実施し、次に、設計者にシステム構築の枠組と部品を提供するために、全体的構造と大域的制御構造、通信プロトコル、同期、データ・アクセスなどの諸機能を支援する制御オブジェクトを整備・提供する必要がある。この時、分散度を考慮しつつ、アプリケーション・ドメインのオブジェクトを、物理的配置に割り当てる設計方法論および記述言語を開発する必要がある。

## 2.2 コミュニケーションはいつ、どのような条件下で成立し、かつ進行するのか？

グループウェアは以下のような技術を開発することにより、コミュニケーションにおける時間／空間分散を克服する手段を我々に提供してきた［９］。すなはち、リアルタイム・グループウェアによる即時性、臨場感の向上。マルチメディアによる適切な情報表現手段の提供。人間間の face-to-face コミュニケーションの前段階におけるコーディネーション・コストの e-mail による低減［１０］。各自の仕事の独立性を保証しつつ必要なコミュニケーションを達成させる非同期コミュニケーション (電子メール) の効果も広く世の中に受け入れられている。

しかし、これらの効果だけでは膨大なネットワーク設備に対する社会的コスト投入の見返りが少なすぎる。コミュニケーションにおける「知性増幅」の道具として、計算機とネットワークを最大限に活用するアプリケーション分野を、我々はさらに開拓する必要がある。

新しい領域を開拓する一つの手掛りは、「会話の背景情報」を明示的に取り扱うことにある。例えばソフトウェ開発においては、ほとんどの会話は膨大な量の文書や設計者やプログラマの知識の一部分に関連している。既存のグループウェアは、頼む、伝える、説明するなどの一つ一つの会話を支援することはできるが、その背景情報となる文書について、今、我々ができることは、関連する文書をスクリーンに表示し操作する程度のことである。ソフトウェア文書の量が膨大であり、記述内容の相互関係も複雑であることを考えにいれると、既存のグループウェアの能力は必ずしも十分でないように思える。大量の情報を参加者間で共有でき、かつフォアグランドで取り扱うことができるような機能をグループウェアは充実すべきであろう。

以上述べた問題点を明確に指摘した論文がある。C. Ellis は［１１］、ソフトウェア・モデリングに関する研究成果に基づいて、グループウェアに対する三つのユーザ視点モデルを定義・検討することにより、上記の問題点に関する今後の研究方向を示唆した。それらは、ユーザが操作可能なオブジェクトの静的記述であるオントロジカルモデル (Ontological Model)、ユーザによって実行されるべき活動を構造的に表現した調整モデル (Coordination Model)、システムとユーザおよびユーザ間のインタフェースであるユーザインタフェースモデルである。

オントロジカルモデルによって、会話の背景情報として必要な静的情報を保持するリポジトリを定義する。誰も全体のことは知らない。多くの場合、各会話参加者はリポジトリの内容の一部を知っており、さらに、それらの情報の一部は他人と共有されている。誰かが各自が保持する情報内容の理解に差があることを認知した時に、コミュニケーションの契機が成立し、情報の均一化の試み (会話) が始まる。会話の内容は、各参加者が持つコンテクストに依存して理解される。構造的コンテキスト (structual context) は「リポジトリの動的状態」を表現する。社会的コンテキスト (social context) や組織コンテキスト (organizational context) は、「会議参加者の動的心理状態と社会的背景情報」をそれぞれ表現する。コンテクストにより会話の流れが制御される。話合いのルールや常識的な手順 (プロセス) を調整モデルが表現する。ユーザ

インタフェースモデルは、関連資料、会話の内容、会議の進行状態などを視覚的手段で会話参加者に提供する。

ここで、考察の範囲を「会話に通じて下される決定事項」に限定しよう。例えば、ソフトウェア開発過程は「決定の積み重ね」であるとみなしうる。各自は各自の仕事の責任範囲において、みずからが下した決定を管理している。ソフトウェア設計者は、設計対象に関する仕様を作成しプログラマに引き渡す。プログラマはその仕様に従ってデータ構造や制御構造を設計しコードを書く。決定 (引き渡される仕様) は、引き渡された時点から、設計者とプログラマの間で共有されることになる。引き渡されたものの内容の理解に関してギャップが存在する時には、いろいろな種類の会話が発生する。また、プロジェクトマネージャとデザイナやプログラマ達と、主に時間資源の管理に関する決定について様々な話題を話合う。

この時、会話の流れを円滑にするために、以下の情報が必要になる。

- 決定事項の全体像の表現
- プロジェクトの目標に対する進捗状況
- ある話題に対する、部分的決定の積み重ねの構造とそれらの決定の際に存在したさまざまの前提条件
- 一つの部分的決定に関する確信度と手間

これらの要請は、相互理解や決定の変更管理支援のために、決定事項をその全体像から詳細レベルにいたる決定の粒度で、系統的に管理できるソフトウェア・リポジトリの必要性を示唆している。リポジトリのモデリングにおいては、決定の時間的特性を考慮に入れるべきである。多くの場合、決定は一時的なものである。時間がたち、状況が変化するにつれて、合意が不合意になり、不合意が合意に変化することはしばしば経験されることである。

決定事項の管理技術を現在のグループウェアの成果に併合することにより、長期間にわたる会話を円滑にし、参加者の間に一体感を作りだす効果を期待できる。

## 2.3 ソフトウェア・プロセス・モデルでは何を記述すべきか？

ソフトウェア・プロセスに関する研究分野においては、「プロセス記述言語とその実行環境」および「プロセス成熟度」に関する二つの主要な研究成果がある。前者は開発工程設計の道具を与え、後者は、組織の開発能力の判定基準、開発プロセス改善へのガイドラインを与える［１２］。本節では、ソフトウェア・プロセス・モデリングの必要性、有効性に関する筆者の意見をもとに、今後の研究方向を論じる。

ソフトウェア・プロセス記述のねらいの一つは、ソフトウェア設計方法論の形式化にある。これは成功するであろうか？ この議論を進めるにあたり、ソフトウェア設計方法論そのものの有効性と、ソフトウェア設計方法論の形式記述の有効性の問題は注意深く区別する必要がある。ほとんどのソフトウェア設計方法論はソフトウェア開発にあたってのガイドラインを与えているにすぎないので、ソフトウェア・プロセス・モデル (ソフトウェア設計方法論の形式記述) も当然同じ欠点を持つ。ここで我々が議論したいことは、形式的記述そのものの有効性である。

本題に戻ろう。「ソフトウェア設計方法論の形式化が有効であるか？」という問いは、「手続きの抽象は有効であるか？」という問いと同じ意味を持つ。ソフトウェア・プロセス・モデルは、(操作、入力、出力) の３組で表現される「活動」の集合と、それらを構造化する制御構造からなる。(操作１、入力１、出力１) と (操作２、入力２、出力２) に、「型」と「ポリモルフィズム」の概念を適用することにより、抽象表現 (操作、入力、出力) を得ることができる。オブジェクト指向法では、さらに、(操作-x, di, do) と (操作-y, di, do) を ([操作-x, 操作-y], di, do) としてまとめることにより一つの表現単位を得る。しかし、これらはプログラミング言語分野での成果であり、ソフトウェア・プロセス・モデリングに固有のものではない。制御構造に関しては、ソフトウェア・プロセス・モデリングに特有な特徴をいくつか観測できる。その一つに「反復」または「バックトラッキング」がある。ソフトウェア・プロセスにおいては、反復における戻り先は、その非決定性の特徴よりあらかじめ決定することはできない。戻り先は動的に決定される［１３，１４］。

プロセスの実行時変更もソフトウェア・プロセス・モデリングに特有の性質である［１５，１６］。人間の行動は不確実で再現性がないので、プロセス実行時にその実行順序や実行内容が状況に応じてしばしば変更される。すなはち、「非決定性」や「不確定さ」の記述が、プロセス・モデリングに固有の特徴である。

これらの制御構造を記述する見返りとして我々は

何を期待できるのであろうか?この点について、はっきりした見通しを我々は持たないように思える。例えば、ウオータフォール・モデルにおいてもオブジェクト指向開発法においても、「反復」は必ず出現する。繰り返しは、前者では悪であり、後者では善である違いがあるだけにすぎない。ソフトウェア・プロセスを形式的に記述することは、ウォータフォール・モデルの欠点を定量的に評価することを可能にするのだろうか? オブジェクト指向ソフトウェア開発におけるプロトタイピング・アプローチに対して、有用で具体的なガイドラインを提供できるのだろうか? 柔軟な標準開発工程の設計を助けてくれるのであろうか? そのような効果を期待できないとすると、形式化は、実世界の活動を別の世界の言葉で単に表現しなおしているにすぎない。

問題の起源に立ち帰る必要がある。人間は完全なものを一度には完成できないので、繰り返しが発生する。その度合は、ソフトウェアの規模が増大するにつれて大きくなる。また、人間は機械ではないので、その振舞いを予測し再現するのが困難である。ソフトウェア・プロセス・モデリングにおいては、これらの人間要因を考慮に入れる必要がある。

我々のアプローチでは、手順を書き下すことではなく、仕事のコンテキストを定義することに力点を置く。試行錯誤は自然なものであるとして、達成目標 (品質基準)、犯してはならない制約や規則、他人との仕事のつながりとその優先度などを各自の作業環境のまわりに配置する。品質基準はオブジェクトベース中のオブジェクトの属性とする。制約、規則、他人との仕事の関係は作業インタフェースの属性とする。上記の事柄に違反した際の調整支援は今後の重要な研究課題である。

プロセス成熟度に関する関する研究は、実世界における様々な開発プロセスを改善するための有用なガイドラインを提供しうるだろうか? F.Cattaneo は論文［１７］で、CMMの適用が必ずしも現実の改善にはつながりないことを指摘している。何が原因なのであろうか?

いくつかの可能性のうち、ここでは、問題発生源の一つとして、個人の活動とチームの目標の間の「調整」の問題を吟味してみよう。例えば、他人のやった仕事を引き継ぐ。他人に仕事を引き渡す。社内やプロジェクト固有の規則を順守する。一定の品質が保証されたプロダクトを生成するなどの例がある。最も重要なことは、これらすべてのことを納期にあわせて、限られた時間資源で実行しなければならないことである。

個人の目標をプロジェクトの目標の間には大きなひらきがある。前者は、「自身の能力の範囲内で納得しながら良い仕事を楽にする」という自己中心の仕事のタイプを容認し、後者は、「限られた時間範囲内にすべての仕事を終了する」という自己犠牲、献身型の仕事のタイプを要求する。双方を同時に達成できる人もいるがそれは稀な例である。われわれは、そのようなノウハウの共有を可能にする理論とシステムを、事例推論とモデル推論を利用して開発したことがある［１８，１９］。しかし、膨大なコストを必要とするので実用化の見通しはない。個人の目標とプロジェクト・チームの目標をつなぐことができる調整支援のためのプロセス・モデルが必要である。

## 2.4 2章のまとめ

本章では、ソフトウェア・エンジニアリング活動を対象として、計算機ネットワークを介した共同作業支援のためのモデルを定義し、環境を構築するにあたっての基本方針と原理を述べた。我々の目標は、以下の四つの特性で特徴づけられる「環境の快適性の向上」にある。

明解性: 中間成果物の評価が可能であり、その手順が明確かつ具体的である設計方法論を利用できること

独立性: 必要なコミュニケーションを保証しつつも、個人の作業の独立性が保持できること

円滑性: チームの構成員間のコミュニケーションが円滑であること

携帯性: 活動の場から、必要な情報に迅速にアクセスできること、さらには活動の場にそれを支える情報が追随できること

次に、今後解決されるべき研究課題を洗いだす目的で、ソフトウェア設計方法論、CSCW, ソフトウェア・プロセス・モデリングの分野における研究現状を吟味した。その結果、以下のような課題を認識した。

- ネットワークにまたがる計算構造を前提として、規模に応じたソフトウェア・アーキテクチャ・クラスを整備すること
- 長期間にわたる会話を円滑にし一体感を生みだすために、リポジトリ管理技術 (特に決定事項の

管理技術) を既存のグループウェアの研究成果に融合させること

- 個人の目標とプロジェクト・チームの目標をつなぐことができる調整支援を目的としたプロセス・モデルを開発すること
- 予測不可能性、非再現性、不完全性などの人間要因を上記のモデルや環境の開発にあたって十分に考慮すること

# 3 「自在」プロジェクト

本章では、1章、2章における問題提起に一つの解を与える目的で進めている、落水研究室における「自在プロジェクト」[20] の進捗状況を紹介する。

まず、決定事項の管理を基にしたソフトウェア・プロセス実行支援とコミュニケーション支援の統合法を検討する。次に共同作業支援の参照モデルである「CSCSD モデル」(Computer Supported Cooperative Software Development model) を提案する。3番目に、CSCSD モデルに基づくプロトタイプ「自在」の開発現状を以下の4つのサブシステムに関して紹介する。

- 「携帯性向上」のための CSCSD サーバ：インタネットワーキング、移動計算環境、遍在情報環境などの新しい計算環境下において、データの非均一性を吸収することにより、個人の計算環境の携帯性を増加させる。プロトタイプ「飛翔」を開発中である。
- 「独立性を保証する」分散サーバー：「分散作業空間オブジェクト」や「自律仲裁オブジェクト」などの制御オブジェクトにより、共有情報の管理を支援する。モデルとプロトタイプ「群舞」をすでに開発した。
- 「円滑性を保証する」グループウェア・ベース：討議の進捗と筋道を記録することにより、決定事項の管理を支援する。決定の粒度に基づいて構成された3層スキーマを定義し、プロトタイプシステム「栞」を開発した。
- 「探求的学習支援」のためのアクティブ・チャネル：上記サーバ上で稼働する新しいアプリケーション。プロトタイプ「探求」を開発中である。

4番目に、プロトタイプ「自在」のアーキテクチャ設計に関して基本的事項を検討する。最後に、ネットワークを介して実施したソフトウェア・モデリングに関する共同作業のプロトコル解析結果を紹介する。実験の目的は、人間系の振舞いを観測することにより、快適性の既存の要因を評価し、また、新たな要因を発見することにある。自在のフィールドテスト環境である「分散プロジェクト」の活動内容も紹介する。

## 3.1 CSCSD モデル

我々の共同活動の主要な要素は何であろうか？それらはどのように関連して共同活動を成立させているのだろうか？少なくとも、プロダクト、コミュニケーション、プロセス、人間の四つを挙げることができる。本節における考察の目標は、これらそれぞれの要素が共同作業で果たす役割を吟味し、要素間の相互関係を定義することである。考察の結果として、CSCSD モデルと名付けた共同作業の参照モデルを提案する。

著者はすでに、ソフトウェア・エンジニアリング諸活動を支援するためにソフトウェア・リポジトリで管理されるべき情報を検討し、( コミュニケーション支援、プロセス管理支援、プロダクト管理支援 )x( 個人レベル、チームレベル、プロジェクトレベル) の9つの分類した [21]。本論文では、それに加えて、第2章で議論した以下の要因を加味した考察を行なう。

- 一体感を生みだし、かみくだく過程を支援することの必要性
- その振舞いは予測しにくく、完全なものを一度には完成できず、その過程には、一般に再現性がないという人間の能力の限界への配慮
- 個人 (自身の能力の範囲内で仕事を進める) とプロジェクト (限られた期限内に成果物を達成する) 間の目標のひらき

これらの要請を満たす筋道の通った解が存在するのであろうか？ もし存在すれば、それが CSCSD モデル構築の基礎を与える。その解の実行にあたって、計算機の特徴を活用できる場面は何であろうか？ これは「自在」設計の基礎を与える。

### 3.1.1 決定事項の管理に基づくプロセスとコミュニケーション支援の統合

ソフトウェア・プロセスの実行時に下される様々な決定とプロダクトやコミュニケーションとの関係

を明らかにしつつ、特にコミュニケーションを通じた決定事項の記録・管理法について考察する。

### 決定とは

ソフトウェア開発における決定事項は2種類ある。一つは、設計者やプログラマによって作成され、検査チームによって検証される中間生成物や製品である。もう一つは、そのような中間生成物や製品を作成していく過程でなされる決定、すなはちいくつかの代替案の中からの一つの案を選択する過程でなされる決定である。前者は、後者が複数個集まって構造化されたものである。

後者の型の決定(デザイン・ラッショナレ)は伝達コストの安い会話によってなされる傾向がある。そこで、ほとんどの研究者は、デザイン・ラッショナレを管理するのに、会話内容そのものをその流れに沿って記録すれば良いという立場を取る[22]。しかし、これは冗長である。なぜならば、このような会話は、内容をかみくだいて説明したり(パラフレージング)、正確に内容を理解してもらうための質疑応答を数多く含むからである。

### 何を記録すべきか

コミュニケーションの円滑化のためには、参加者各自が、現在の話合いの内容が全体目標に対してどのような位置づけにあるのかを理解できることが重要である。また、「完全なものを一度には作成できない」という人間の特性により決定事項はしばしば変更される。そこで、決定は多くの場合一時的であり、「完全な決定」はありえない。決定の変更とその波及の解析が容易に行なえる記録スキーマが必要である。

決定事項の管理とは、中間生成物とその依存関係に加えて、上記の内容を記録・管理することである。両者の型の決定をどのように記録管理すべきであろうか? これがCSCSDモデル定義にあたっての主要な論点である。

ここで、コミュニケーションを円滑にする別の要因、「かみくだく過程の支援」や「一体感の形成」に言及しておく必要がある。共同問題解決活動に携わる場合、結論にいたる速度、質が人によって異なる。早く達成した人が、ゆっくり、かみくだくように、要点を「他人の能力」を勘案しつつ説明する状況が話合いでは良く出現する。これを「かみくだく過程」と呼ぶことにする。かみくだく過程は、話者相互間で、理解の差や決定内容に関するくい違いが自覚され、それをなくそうとする努力がなされている場合にも生じる。

かみくだく過程の会話を記録することは、その内容が人の経験・知識・体調等に依存するものであり再現性がないので、あまり意義はないと思われる。もちろん、かみくだく過程を支援することは重要であり、参加者の間に「一体感」を生み出し、進行状況や次の議論の目的(決定事項と未決事項)の共有を助ける。かみくだく過程の支援はグループウェアやユーザインタフェースの役割である。

### 中間生成物間の粗粒度レベルの依存関係を統合の土台とすることについて

分析・設計活動は、ある中間生成物を入力して、既存の成果物、標準規約、品質基準等を参照しつつ、ツールによる変換や人間の思考によって出力となる中間生成物や製品を生成する。すなはち、中間生成物や製品の間には入出力に関する依存関係が存在する。

粗粒度レベルの依存関係は、仕様書や設計書などの文書間の入出力依存関係を定義する。中間生成物/製品間の粗粒度レベルの依存関係に実行順序を付加することにより、ソフトウェア・プロセスを定義できる。細粒度レベルの依存関係は、それら文書中の項目間の「利用する/される」または「参照する/される」という依存関係を定義し、設計根拠を理解するための基礎を与える。我々は、中間生成物間の粗粒度レベルの依存関係をソフトウェア・プロセス・モデルの定義とコミュニケーションの記録を統合する基礎として採用する。

もちろん、中間生成物間の粗粒度レベルの依存関係が上記の機能実現の土台として必要十分であるとはいえない。しかし、タスクの実行順序の設計にあたって制約を与え、会話の発生順序に関する概略を与えている意味において、統合の主要な土台の一つである。

### 粗粒度レベルの依存関係を基にしたタスクとソフトウェア・プロセスの定義

中間生成物/製品の部分集合を、チーム構成員それぞれに各自の役割に応じて割り当てる。これをタスクと呼ぶ。それぞれのタスクは、例えば、割り当てられた中間生成物や製品の記述に関する様式や規約、作業遂行上の制約、他人との仕事の関係、タスク開始/終了条件、品質基準などの属性を持つ。属性定義の詳細は個々の組織やプロジェクトに依存するので、ここでは、プロジェクト・マネージメント

に有用な、すべてのタスク属性定義に共通となる性質を検討する。少なくとも、以下の機能の実現に必要な情報をタスク属性として定義すべきであると考える。

- 順守すべき制約事項が妥当で明確であること。
- 制約が犯された場合には、システムがそれを検知できること。
- 制約違反に対しては、システムが適切な対応を指示できること
- その処置が他人に悪影響をおよぼす場合には、その調整をシステムが支援できること。

中間生成物／製品間の粗粒度レベルの依存関係に実行順序を付加することにより、タスク間のラティスとしてソフトウェア・プロセスを定義する。プロジェクト全体に割り付けられる時間資源と、能力や容量に応じて各自が必要とする時間資源をバランスさせるところにプロセス定義の難しさがある。

決定事項の記録法について

会話の流れは会話参加者の関心の推移を表現する。そこでは参加者の間で最も関心の高い事項から会話が始まる。その会話の前提条件の欠落に途中で気付き、一時話題がそちらに移ることもある。ある話題について話が一段落した時、続いて話合うべきことが持ち上がる。一度の話し合いでは結論に達せず話を中断することもよくある。この時、別の話題に話が移った後で、突然あることに気付いて話題がもとに戻ることもある。このような会話の流れの中で決定されていく事項をどのように記録すれば良いのだろうか。日記を書くように、会話の流れに沿って記録することは一番容易であるが、決定事項間の因果関係を明示的に表現できない欠点がある。

我々は、話合いの途中で、そこまでの話を整理し、解決された問題とそうでないものを整理した上で次に話し合うべきことを決定するような場面をよく経験する。この時、我々は以下のような情報を整理している。

- すでに決定されたことと、これから決定されるべきことに対する全体像
- 全体像に対する、現状の位置づけ
- 最終目標に対する進捗状況
- 次に話し合うべき話題

上記の事柄は記録の対象とすべきであろう。ところで、議論が進むにつれてそのような整理結果は変化することを考慮に入れる必要がある。また、いったん決定したことが不完全であったことに後になって気付くこともしばしばある。討議結果の再構成や蒸し返しを支援できる情報も記録の対象に入れるべきである。

- ある話題に対する、部分決定の積み重ねの構造とそれらの決定にあたって必要なさまざまの前提条件
- 一つの部分的決定に関する確信度と手間

さらに、話し合いの話題はソフトウェア・オブジェクトに起因しているので、

- 中間生成物や中間生成物間の依存関係との関連

を記録する必要性は明白である。ここで、討議空間 (最上層)、討議プロセス (中間層)、討議の型 (最下層) からなる 3 層記録スキーマを導入する。

- 決定事項間の因果関係を討議空間で表現する。その役割は、決定事項の全体像と最終目標に対する現在の議論の適切な位置づけを我々に提供することにある。
- ある話題に対する一連の会話を討議プロセスで管理する。その役割は、討議の型を話題対応に討議プロセスとしてまとめることにより、ある話題に対する、部分決定の積み重ねの構造と、それらの決定にあたって必要なさまざまの前提条件を管理することにある。
- 会話中の一連の発言を伝達・整合、決定、創造などの討議の型で管理する [２３]。ソフトウェア開発プロセス時に発生するほとんどの会話は以下の三つの型に整理することができる。良く知っている人がそうでない人に決定事項や知識を「伝達」する。お互いの言い分を聞きながら、いくつかの代替案の中から一つの案を「選択・決定」する。部分的な知識を持つ人々が、それらを統合して新しい情報を「創造」する。討議の型の役割は、一つの部分的決定に関する確信度と手間を記録することにある。伝達・整合、決定、創造などの討議の型はいくつかの状態を持ち、どの状態で会話が中断・終了したかにより、

確信度や完遂度を判定できる。時間属性や発言の回数は決定に要したコストを表現できる。

上記スキーマを持つリポジトリをグループウェア・ベースと呼ぶ。

ソフトウェア・プロセスの実行と
発生するコミュニケーションの
関係について

ところで、コミュニケーションを通じて決定される事柄の間の因果関係と、タスク間の時間的順序関係は同じ構造を持つのであろうか？ 討議空間の構造は、後の例に述べるように、話題が状態に依存して発生することがあるので、タスクの実行の構造と必ずしも一致しない。そこで、「決定事項間の因果関係」と「タスクの実行順序関係」はそれぞれ独立に管理することにする。

例題

討議の型とタスクの対応

タスクに割り当てられた人間が複数人である場合と、タスク間で中間生成物が直接または間接に共有される場合にコミュニケーションが発生する。発生するコミュニケーションにはいくつかの種類がある。例えば以下の例は代表的である。

- 複数人で一つの仕事を共同で実施する場合に、それぞれが持つ専門知識を交換するため、代替案の中から解候補を選択するため、スキルや経験の差を補うため (かみくだく過程) にコミュニケーションは発生する
- タスク間で中間生成物が共有されている場合に、生成物の伝達のため、伝達内容に関する合意形成のため、中間生成物の不完全さに起因する誤りを検知・修正するためにコミュニケーションが発生する

討議の型のインスタンスはタスクまたはタスク間の接続に直接関係する。

討議プロセスや討議空間と
タスクの実行構造の対応

機能仕様書は複数の機能要求から成り立っている。設計段階において、それらの機能の内、一部は性能要求を満たすため早いプロセッサに割り付けられ、残りの機能は遅いプロセッサに割り付けられる。それぞれはまとめられてモジュールになる。それらのモジュール間のインタフェースが決定され、大域データと各モジュールのデータ構造が決定され、各モジュールが実装される。テスト時にロードモジュールが十分な性能を出せなかったとしよう。すでになされた以下の 2 種類の決定を変更しなければならない。

- 粗粒度レベルの決定：機能仕様書、性能仕様書、プロセッサの選択、二つのモジュール、ロードモジュール。
- 細粒度レベルの決定：ある機能のプロセッサへの割りつけ、割りつけられた機能のモジュールへの統合、モジュール間インタフェース・データの選択、データ構造とアルゴリズムの選択。

プロセッサの選択は妥当であったか？ 機能の振り分けは妥当であったか？データやアルゴリズムの選択は妥当であったか？ 等の決定事項を再検討しなければならない。

この結果、さまざまの仕様書やプログラムが変更しなければならない。これに対しては、独立した討議プロセスを起動し、サブ討議プロセスで関連する仕様書やプログラムに関する決定内容の変更を検討し、変更結果をその理由と共に追加記録するのが良い。

この例のように、予測不能は異常状態が出現した場合には、ソフトウェア・プロセスとは独立した討議プロセスを起動する必要がある。一般に、「決定事項間の因果関係」と「タスクの実行順序関係」は一般には異なるものである。

### 3.1.2 共同作業のモデル：CSCSD モデル

CSCSD モデルにおいては、以下の二つをモデル化の中心話題とする。

- 中間生成物の粗粒度の依存関係をもとにした、各自の責任範囲を明示するタスクの管理、タスク間の時間的実行順序 (ソフトウェア・プロセス) の管理
- 細粒度の依存関係を基にした決定事項とその間の因果関係 の管理 (グループウェアベース)

上記の諸要素を階層構造として以下のように構造化する。

- 最下層には CSCSD サーバを置く。ネットワーク・トランスペアレントなデータのアクセスを可能にし、データの非均一性を吸収する。

- 下から 2 番目の層では、中間生成物間の依存関係を管理する。上層の分散サーバを構築する際、作業のガイドラインや各自の作業に関する制約を定義し、またグループウェア・ベースに対してはコミュニケーション記録にあたって、会話内容がどの中間生成物やその依存関係に関与するのかを定義する土台を与える。

- その上の層には分散サーバ、グループウェア・ベースを置く。分散サーバはさらにプロセス・サーバと分散作業空間からなる。プロセス・サーバは、タスクとその実行順序、消費可能資源等を定義する。分散作業空間は、作業分担と責任の範囲を明確にし共有情報の管理を行なう。グループウェア・ベースは、会話による決定事項とその間の因果関係を保持する。

- その上の層 (上から 2 番目の層) には、CASE tool やグループウェアを置く。中間生成物の変換活動、合意形成や伝達・決定などのコミュニケーションを支援する。

- 最上層はユーザインタフェース管理システムを置く。ユーザインタフェースには、分散サーバ、グループウェア・ベース、プロセス・サーバの状態が表示されると共に、Ellis が指摘した各種情報、構造的コンテキスト、社会コンテキスト、組織コンテキストが必要に応じて表示され操作される。

## 3.2 自在プロトタイプの研究現状

本節では、自在プロジェクトにおけるプロトタイプ開発の成果をまとめる。われわれは、現在、第 1 章で設定した目標と、第 3 章で導入した CSCSD モデルに基づきいくつかのプロトタイプを開発中である。モデル中で定義された諸機能の実行にあたって計算機パワーを活用できる部分は何であろうか。また人間がやるべき部分と計算機化されるべき部分のインタフェースをどのように設定するのが適切であろうか。これらの問題を CSCSD モデルの各層毎にまず検討する。

- CSCSD サーバ (最下層)…この層の機能は可能なかぎり自動化することが望ましい。また、その機能は利用者に対して隠蔽されることが望ましい。

- 中間生成物間の粗粒度の依存関係の管理 (第 4 層)…作業順序に関するスキーマの表現およびデータアクセス (生成・削除・変更) の管理は、データの一貫性を保持するためにデータベース化することが望ましい。

- 分散サーバとグループウェア・ベース (第 3 層)…ソフトウェア・エージェント (制御オブジェクト) を活用する手段を工夫すべきである。また、グループウェア・ベースに関しては、決定の変更と波及の解析に対応できる記録構造をスキーマとして計算機内部に表現するべきである。

- ツールキット層 (第 2 層)…CASE tool やグループウェアに対するツール統合機構の支援が、作業の連続性を保証するために必要である。

- ユーザインタフェース (最上層)…構造的コンテキスト、社会コンテキスト、組織コンテキストなどの表示のためにはグラフィックスの活用が必然であり、マルチウィンドウに替わる新しいメタファの定義とそれを支援する UIMS の開発が必要である。

### 3.2.1 携帯性向上のための CSCSD サーバ

インタネットワーキング、移動計算環境、遍在情報環境などの、新しい計算環境に対する基盤が発展しつつある。これら一群の新しい計算機環境は、どのような新しい可能性を我々にもたらすのであろうか。一口でいえば、「活動の場から、必要な情報に迅速にアクセスできること、さらには活動の場にそれを支える情報が追随できること」であろう。これらの新しい計算機環境の特徴の一つは非均一性である。自在プロジェクトの一貫として開発中の CSCSD サーバの目的は、インターネットと移動体通信技術、移動計算環境と遍在計算環境を活用できるインフラストラクチャを提供することである。

このようなサーバを構築するにあたって考慮すべき点がいくつかある。例えば、データの非均一性や操作の非同期性への対応がある。Gio Wiederhold は、この問題に対して、従来の Client-Server モデルを、Client と Server の中間に Mediator をおく Mediatted Architecture に拡張することを提案している [２４]。Mediator は、適切な情報資源へのアクセス、データ選択、フォーマット変換、データを共通抽象レベルで整理すること、異なる情報資源からの情報を統合

すること、アプリケーションの目的に応じたメタ情報を準備することなどを支援する。

このようなソフトウェアは一般にミドルウェアと呼ばれ、すでに、CORBA( Common Object Request Broker Architecture)、DOE( Distributed Objects Everywhere)、ILE( Inter-Language Unification )、KQML(Knowledge Query and Manipulation Language)、OLE( Object Linking and Embedding)、OpenDoc(Open Document Exchange )、PDES( Product Data Exchange using STEP) などの商品が出ており、今後標準化が推進される (http://isx.com/pub/I3)。

また、S. Heiler は、大規模分散システムにおける相互互換性 ( interoperability) の問題を指摘している［２５］。非均一なシステム構成でサービスやデータを交換するために、通常の、メッセージパッシングプロトコル、手続き名、エラーコード、引数の型などの合意に加えて意味的レベルの互換性も保証する必要がある。たとえば、データ名の与え方の問題に関して、２人のデータベース設計者がいるとして、それぞれはドメイン・エキスパートでもあるとする。このとき、同じドメインの同じデータ要素であっても、それに対して同じ名前が与えられる確率は 7%から 18%の間であるという事実を例にあげていて興味深い。

Frank Manola は情報システムをオブジェクト指向アーキテクチャで構築する際の相互互換性の問題を論じている［２６］。古典的な情報システム構築法は、例えば、D.Pascott による DATARUN 方法論［２７］で述べられているように、ほぼ完成の域に近づいたといえる。均一な世界においてはこの手法で十分であるが、大規模分散システムでは、世界はヘテロになり、その間でデータ交換等が保証されなければならない。データベース、共有サービス、アプリケーションの各レベルで、データ表現、オブジェクト・インタフェースなどの相互互換性が保証されるべきである。

開発されるべき CSCSD サーバに対する機能要請をまとめると以下のようになる。

(1) 規模への対応: 規模 (遅延時間) に応じて、大域的制御構造、通信プロトコル、同期、データ・アクセス法などを適切に設定できること。または、基本部品を基に、規模に応じた適切なサーバを容易に構築できること

(2) 非均一性への対応: 適切な情報資源へのアクセス、データ選択、フォーマット変換などが保証されること

(3) 計算機環境への対応: マルチプラットフォーム上で稼働可能なこと

(4) アーキテクチャの柔軟性の保証: オブジェクト指向アーキテクチャに基づいて設計されていること

### 3.2.2 独立性を保証するための分散サーバ

独立性という目標を達成するための一つの必要条件として、共有情報の管理法を研究している。独立性という用語は以下のような意味合いである。他人との間で不必要なコミュニケーションや相互干渉なしに、各自が自身の仕事を進められること。我々はすでに、作業空間管理者、自律メディエターと名付けた二つの制御オブジェクトの開発に成功した。それらは分散開発における共有データの管理を支援する［２８，２９］。作業空間管理者、自律メディエターの主要な特徴は以下の通りである。

1. 作業空間管理者オブジェクトと中間成果物オブジェクトは、ソフトウェア開発者各自の仕事の責任範囲を規定し、共有データの管理を支援する
2. 自律メディエター・オブジェクトは、共有データの変更にともなって発生するソフトウェア開発者間のネゴシエーションを支援する
3. それぞれのオブジェクトは, 対応するメタ・オブジェクトを持ち、状況に応じて適切な振舞いを動的に選択することを可能にする

これらのオブジェクトを利用することにより、ソフトウェア技術者は、彼の責任範囲に直接関係する知識と、彼とデータを共有する人との関連のみに注意を払って仕事を進めることができる。我々の環境はまた、データ共有に関する方針を協調的かつ柔軟に変更することを支援する。

次の研究段階では、作業空間管理者オブジェクト、自律メディエター・オブジェクトに一定のレベルの知性を付加することにあり、トリオ環境［３０，３１］における研究成果を適用することを予定している。トリオ環境における研究成果は、その振舞いが時制論理で仕様化されたオブジェクトの並行制御とコミュニケーションの問題を検討することでソフトウェアプロセスモデルに拡張可能であると思われる。

例えばトリオ環境では、オブジェクトの振舞いは以下のセマンティックスを表現するで形式化される。「事象Aの発生に伴ない、事象BがX秒以内に発生しなければならない」。

トリオ環境の成果をソフトウェアプロセスモデルに適用する場合には、検討されるべき点が二つある。まず第一は、トリオモデルではWFFは「単一の現在時刻」を基準にして表現される。ソフトウェア・プロセス・モデリングにおいては、それぞれのオブジェクトが固有の現在時間 (仕事の開始時間) を事前条件と共に持つように定義する方が自然である。また、ソフトウェア・プロセスの実行中に生じるオブジェクト間のコミュニケーションも定義する必要がある。これは、並行実行制御とコミュニケーションの問題である。二番目は、ソフトウェア・プロセス・モデリングにおいては、「…時間以内」に「必ず…しなければならない」という表現はプロジェクト管理者の単なる期待であることにある。人間の行動は予測し難く、また再現性がない。

まとめると、ソフトウェア・プロセス・モデリングにおいては以下のような期待と現実の喰い違いに関するセマンティックスを形式表現する必要がある。「もし、あるオブジェクトが割り当てられた制約に違反した場合、他のオブジェクトがそれを検知できなければならない。さらに、もし可能なら、人間の助けを得つつも、オブジェクトが互いにネゴシエーションすることでその解を発見できなければならない」。

### 3.2.3 円滑性を提供するためのグループウェアベース

円滑性向上の必要条件の一つである「チーム構成員間のコミュニケーションの円滑化」を目標として、チームにおける「決定事項の管理の支援法」を研究している［３２］。我々はすでに、討議の進捗状況と討議内容の筋道を記述するフレームワークと図式表現を提案した。我々が提案したスキーマの目標は、長期間の議論によって生成される決定事項を、決定の全体像の把握や決定の変更が容易であるように管理することにある。記録スキーマは討議空間、討議プロセス、討議の型の 3 層からなる階層構造で構成される［２３］。

- 討議空間は、決定事項の全体像とグループの目標に対する完遂度を表現するために、話題間の因果関係を記録する。
- 討議プロセスは、部分的決定の積み重ねの構造と決定に関するさまざまな前提条件を記録することにより、ある話題に対する討議の進捗状況を表現する。
- 討議の型は、討議プロセスを構成する特定の会話における発言を伝達・整合、決定、創造などの討議の型としてまとめて管理する。また、会話による部分的決定の確信度と手間を記録する。

以上述べてきたように、我々のスキーマは、決定事項を抽象レベルから詳細レベルにわたって系統的に管理することで、決定事項の周知と変更管理を支援する。上記のスキーマによって管理されるデータベースをグループウェアベースと呼ぶ。

記録スキーマの設計にあたってとくに以下の点に留意した。「さまざまな粒度の決定がなされるとき、「合意」というものは、多くの場合一時的であり、状況が変化すれば、合意が不合意になり、不合意が合意に変化する」。

我々はすでに、上記のモデルに基づいてプロトタイプ・システム「栞」を開発済みである［３３］。プロトタイプは、メーリングリストを利用した電子会議を対象として、討議参加者間のさまざまな意見の喰い違いの調整を支援し、また、決定事項の管理の不十分さに起因する冗長な会話の発生を減少させる目的で開発した。

### 3.2.4 探求的学習を支援するアクティブチャネル

自在を利用した先進的ソフトウェア開発環境は、良いソフトウェアを開発するのに必要な日常活動；news や e-mail による情報の発信と獲得、内外の研究者とのディスカション、電子会議による合意形成・意思決定・共同設計作業、研究成果のリアルタイムな発表などの活動をネットワークを介して実施することも支援する［３４］。この目的のために開発中のアプリケーションが、遍在情報に対するアクティブ・チャネルである。

今後、ネットワークに点在する膨大な量のデータの中から必要な情報のみを抽出・利用する情報フィルタリング技術を基にした情報検索技術の革新がおこるだろう。すでに WWW の普及により、ネットワークの各ノードに様々な情報が蓄積されつつある。このような情報を仮に遍在情報と呼ぼう。各地に散在する遍在情報の中から、情報獲得に関してごく限られた知識しか持ち合わせない人が、計算機の助けを

借りて問題を解決する探求的学習の支援は今後の大事なアプリケーションの一つであると思われる。探求的学習を支援するツールをアクティブ・チャネルと呼ぶ。ここでチャネルとは情報の検索要求に応じて動的に張られる情報検索・フィルタリングのためのリンクを意味し、アクティブとは、一回の検索ごとに検索やフィルタリングに有用な情報が増加することを指す。

探求的学習に関する基本的な機能要請は Denning によって議論されている［３５］。Denning は CSCW のタスクドメインを「手段」、「情報交換の媒体」、「探求的学習の支援」の 3 つに分け、各ドメインにおけるソフトウェア・エージェントが果たすべき役割を論じている。

**(1)** ここで、「手段」とは、既知のタスク・ドメインで、ある特定のゴールを達成する (出力を得る) 手段として CSCW を利用することである。

**(2)** 何が出力として得られるかは必ずしも予知できないが、それを達成するための手段を知っている場合には、CSCW は「情報交換の媒体」として利用される。ゴールはシステム利用者間のインタラクションによって達成される。

**(3)** 利用者はタスク・ドメインやある話題について、ごく限られた知識しか持ち合わせていず、かつ、特別な出力を持つわけではない。

アクティブ・チャネルについては、「アクティブネス」に関する機能の設計と実現を現在試作中である。

## 3.3 自在プロトタイプのアーキテクチャ設計

３．１．２節で定義した CSCSD モデルの各機能を計算機環境への割り当てを検討しつつ、自在プロトタイプの全体的構造と大域的制御構造等を考察する。規模に応じたアーキテクチャ・クラスの分類はまだ行なっていないので、ここでは日本国内を対象としてインターネット環境の上に構築されるシステムを仮定する。

### 3.3.1 各種サーバとデータベースの配置

- **CSCSD サーバの配置:** CSCSD サーバの配置法としては三つの可能性がある。
  1. 各データベース毎に CSCSD サーバを置く方式
  2. 情報検索がいくつかのデータベースにまたがることに対応するため独立した CSCSD サーバをシステムに配置する方式
  3. 各ユーザが一つ CSCSD サーバを持つ方式

  それぞれの方式の長短を検討する。方式 (1) のメリットは、CSCSD サーバを各データベース管理システムに依存した構成にカスタマイズすることが可能になり、各データベースへのアクセス効率はあがる。しかし、通常は、一つの検索要求はいくつかのデータベースにまたがることが多いので、CSCSD サーバ間の交信のオーバヘッド増が問題となり、必ずしも全体的効率が上るとは限らない。また、データの一貫性保持もネットワークにわたって考慮する必要がある。

  次に、方式 (2) を検討する。ただ一つの CSCSD サーバをシステムに配置するのは負荷の面で現実案ではない。何を基準にして複数個配置するかが問題である。ソフトウェア開発では、かなりの長期間にわたって同じデータが参照更新され続けることを考慮に入れると、プロジェクト対応に置くのが妥当であると思われる。この時、性能を考慮して CSCSD サーバにはキャッシュを設け、一貫性制御は CSCSD サーバの責任とする必要がある。

  ユーザ対応に配置する方式のメリットはただ一つである。CSCSD サーバのキャッシュ内にユーザに関係するデータのみが置かれるので、ネットワークにまたがったデータアクセスが少なくなる。

- **中間生成物間の依存関係の管理機能の配置:** これはプロジェクト毎に置くのが妥当であろう。しかし、独立したサーバを設けるか、CSCSD サーバの作業領域に置くかで選択の余地がある。中間生成物間の依存関係はかなり静的な情報であり、プロジェクト管理者によってときどき更新されることを考慮にいれると、独立したサーバを設け、プロジェクト管理者の近くに配置するのが望ましいように思える。

- **分散サーバとグループウェア・ベースの配置:** 分散サーバはその役割から明らかなように、論理的には各自対応に置かれる。物理的配置も利用者のレスポンスタイムに関する快適性を保証するためにはユーザ対応に配置した方が良いと思われる。グループウェア・ベースに関しては、討

論の全体像を把握する場合と、例えば、ソフトウェアモデリングやレビューなどのようにヘビーなインタラクションが行なわれる場合は分けて考える必要がある。前者の場合は例えばウェブサーバを介して参照できれば十分であろう。後者の場合は、討議プロセス対応に棄サーバを配置するのが効率の面で望ましいと判断する。

- ツールキットの配置: 個人の環境内に設定される
- ユーザインタフェースに関する問題: UIMS は当然個人の環境内に配置することになる。コンテキスト情報の配置については表示速度の面からその配置を十分考慮する必要がある。structual context については、CSCSD サーバ、分散サーバの配置と連動して決定する必要がある。social context や organizational context に関する情報はグループウェアベースの配置と連動して決定する必要がある。いずれにしても、この機能に関連するデータの寿命は短く、また即時性が強く要求される。今後の検討課題である。

### 3.3.2 自在プロトタイプのオブジェクト指向アーキテクチャ

アーキテクチャの柔軟性を増加させるため、自在プロトタイプはオブジェクト指向アーキテクチャで実現する予定である。この課題については、現状は、アーキテクチャの構成要素として、workspace-manager と autonomous-mediator を開発したにすぎない。今後の課題である。

## 3.4 プロトコル解析による人間系の振舞いの理解

本論文においては、計算機環境の携帯性の向上、開発対象とその手段の明確化、作業の独立性の保証、コミュニケーションの円滑化で特徴づけられる計算機環境の「快適性の追求」を議論してきた。しかし、定義されたモデルや開発されたプロトタイプ環境の良さを評価するためには、評価のための手段・尺度が必要である。この手段としてプロトコル解析が有用であると思われる。我々はプロトコル解析実験の目的は次の二つに設定する。

- 「快適性」の別の評価尺度を発見する
- 「独立性」や「円滑性」に対する評価手段を開発する

前者は、携帯性、明確性、独立性、円滑性を含め、快適性の要因を再吟味することを意味しており、後者はそれらの評価手段を開発することを意味している。自在プロジェクトでは、現在、主にコミュニケーションの円滑化に関連して、既存のグループウェアがどのような問題点をはらんでいるのかを定量的に把握しようとする試みが行なわれている［３６］。

## 3.5 自在のフィールドテスト環境「分散プロジェクト」

「分散」プロジェクトは、北陸先端大、奈良先端大、東工大などの複数の大学や、PFU を始めとするいくつかの企業が参加した共同プロジェクトである［３７］。「分散」プロジェクトの目的は、研究室における基礎的な研究成果をネットワーク上で評価する場を設けることである。北陸先端大は、「自在」プロジェクトのフィールドテスト環境としてして位置づけている。分散プロジェクトでは、以下のような共同実験を実施してきた。

- 遠隔プレゼンテーション、遠隔形式仕様レビュー、遠隔共同プログラミングなどを Internet、ATM などのネットワーク設備上で、既存のマルチキャスト・ツールを利用して実施してきた。実験の目的は、既存のマルチキャスト・ツールやネットワーク設備に関する問題点を発見することにあった。すでに第一期の一連の実験を終え、ネットワーク層における遅延の問題、人間系におけるコミュニケーション・プロトコルの問題等、今後の研究・技術開発に有用な知見が得らた。
- またネットワークを介した共同作業の阻害要因を把握するためのプロトコル解析実験も実施した。

分散プロジェクトにおける JAIST の次の目標は、CSCSD サーバ「飛翔」の運用と改良、分散サーバ「群舞」の運用と改良、グループウェア・ベース「棄」の運用と改良である。

## 3.6 3 章のまとめ

決定事項の管理とその生成・変更の支援をモデル化の中心として、下記のような機能階層からなる共同作業の参照モデル CSCSD モデルを提案した

ユーザインタフェース (最上層): 構造的コンテキスト、社会コンテキスト、組織コンテキストなど

を表示し、共同作業の参加者が、「一体感」を形成することを補助する。

**ツールキット層 (第 2 層):** 作業の自動化の度合を高めるCASE toolや、「かみくだく過程を支援するグループウェアを置く。また、アクティブ・チャネルのような新しいアプリケーションもこの層におかれる。ツール統合機構により作業の連続性を保証する。

**分散サーバとグループウェア・ベース (第 3 層):** 作業空間の管理と変更管理の支援を、各自の作業の独立性を保証する観点から支援する分散サーバと、決定事項の参照・変更・波及解析等を支援するグループウェア・ベースを置き、コミュニケーションの円滑化をはかる。

**中間生成物間の依存関係の管理 (第 4 層):** 中間生成物間の依存関係を管理する。また、それを基にした、作業順序に関するスキーマ (プロセスモデル)、およびそのインスタンスの生成・削除・変更に関する一貫性制御をおこなう。

**CSCSD サーバ (最下層):** 規模への対応、非均一性への対応、計算機環境への対応などを支援する。

次に、現在進行中の「自在」プロトタイプを紹介した。また、「自在」プロトタイプのアーキテクチャ構成についても簡単な考察を行なった。

## 4 まとめと今後の課題

本論文では、ソフトウェア・エンジニアリング活動を対象として、計算機ネットワークを介した共同作業支援のためのモデルを定義し、環境を構築するにあたっての基本方針と原理を述べた。我々の目標は、明解性、独立性、円滑性、携帯性の四つの特性で特徴づけられる、環境の快適性の向上である。その実現のために解決されるべき課題は以下の通りである。

- 規模に応じたソフトウェア・アーキテクチャ・クラスの整備
- リポジトリ管理技術 (特に決定事項の管理技術) とグループウェアの融合
- 調整支援を目的としたプロセス・モデルの開発
- 予測不可能性、非再現性、不完全性などの人間要因の考慮

次に、決定事項の管理に基づく共同作業のモデル、CSCSD モデルを提案した。CSCSD モデルは 6 層の階層構造から成る。それらは、ユーザインタフェース層、ツールキット層、分散サーバとグループウェア・ベース、中間生成物間の依存関係の管理、CSCSD サーバである。

現在、「自在」プロトタイプについては分散サーバおよびグループウェアベースについて初期の研究開発が終了した段階である。新しい計算機環境と開発スタイルに対応できるソフトウェア開発環境の実現という目標に対して、今後の進展させるべき課題は以下の通りである。

- 「規模への対応」を考慮したソフトウェア・アーキテクチャに関する研究について、さらに詳細な調査研究を実施すること
- 「柔軟な」ソフトウェア・アーキテクチャの構成法とオブジェクト指向アーキテクチャに関する研究の現状を調査すること
- 上記 2 つの調査結果ををもとに、ソフトウェア・アーキテクチャのクラスの定義、
- アーキテクチャ構成部品 (制御オブジェクト) の整備方針,
- システムへの統合法に関して考察を深めること
- その後で、いくつかの事例研究を実施し、オブジェクト指向ソフトウェア開発方法論におけるソフトウェア・アーキテクチャ設計の位置づけを明確かつ具体的にすること
- 上記 4 点と並行して、群舞と莽を中心とする自在プロトタイプの第 1 版を早期に完成し、「作業の独立性の保持」、「コミュニケーションの円滑化」に対する効果を評価するためのフィールドテストおよびプロトコル解析を実施すること
- ソフトウェア・プロセスに関する研究に関して、ポリシーや制約の表現法を検討した研究成果をサベイすること。
- また、決定事項の管理の問題に関して、管理の粒度、保護や安全性の度合等に関する現実世界の状況を調査・把握し、モデル化にあたっての具体的目標を設定す

- 上記 3 項目の実験および調査研究の成果を背景に、プロセスとコミュニケーションの統合法に関しての理解を深め、CSCSD モデルを進化させること。
- 自在プロジェクトを世の中で有用であるレベルにまで持ち上げるため、「ソフトウェア共同実験場」の活動の一貫として「自在」コンソーシアムを設立すること

**謝辞**

本報告書に対する Carlo Ghezzi 教授と Stefano Ceri 教授の有益なコメントと討論に対して心より謝意を表します。

**参考文献**

1. 小島:「建築」, イタリア, 新潮社.
2. David Garlan, "Research Directions in Software Architecture", ACM Computing Surveys, Vol.27, No.2, pp.257-261,June 1995.
3. 倉谷、門脇、西山、落水:構造化分析の事例研究(生産管理システム)、Seamail Vol.8 NO. 8-9, pp.3-40, 1994.
4. 倉谷、東田、藤枝、鈴木、落水:オブジェクト指向分析の事例研究(生産管理システム)、Seamail Vol.8 NO. 8-9, pp.41-76, 1994.
5. 落水、東田:「オブジェクト・モデリング」, ジャストシステム, 1995.
6. 落水:「ソフトウェア工学実践の基礎-分析・設計・プログラミング」, 日科技連出版
7. U. Dayal, M. Hsu and R. Ladin: "A Transaction Model for Long-Running Activities", Proc. of 17th International Conference of VLDB, pp.113-122,1991.
8. C.A. Ellis and S.J.Gibbs: "Concurrency Control in Groupware Systems", Proc. of the ACM SIGMOD'89, pp.399-407, May 1989.
9. C.A. Ellis and S.J.Gibbs and G.L.Rein: "Groupware: Some Issues and Experiences", Comm. of the ACM, Vol.34, NO.1, pp.38-58, Jan. 1991.
10. Martha S Feldman: "Constraints on Communication and Electronic Mail", Proc. of CSCW 86, pp.73-90, 1986.
11. Clarence Ellis, Jacques Wainer: "A Conceptual Model of Groupware", Proc. of CSCW 94, pp.79-88, 1994.
12. 落水:「ソフトウェア・プロセスに関する研究の概要」, 情報処理,Vol.36, NO.5, pp.379-391, 1995.
13. A.Ohki, K.Ochimizu: "Process Programming with Prolog", Proc. of 4th ISPW, pp.118-121, 1988.
14. K.Kishida, T.Katayama, M.Matsuo, I.Miyamoto, K.Ochimizu, N.Saito, J.H.Sayler, K.Torii, L.G. Williams: "SDA: A Novel Approach to Software Environment Design and Construction", Proc. of 10th ICSE, pp.69-78, 1988.
15. G.Cugola,E.Di Nitto Frank,C.Ghezzi: "How to deal With Deviations During Process Model Enactment", Proc. of ICSE 17, pp.265-273, April 1995.
16. G.Cugola,E.Di Nitto Frank,A.Fuggetta,C.Ghezzi: "A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems", (submitted to TOSEM).
17. F.Cattaneo,A.Fuggetta,L.Lavazza: "An experience in process assesment", Proc. of 17th ICSE, pp.115-121,1995.
18. K.Ochimizu,T.Yamaguchi: "A Process Oriented Architecture with Knowledge Acquisition and Refinement Mechanisms on Software Process", Proc. of 6th ISPW, pp.145-147, 1991.
19. 山口、樽松、下津、中尾、落水: "事例に基づく推論とモデル推論の統合に基づく知識獲得支援システム(2)-ソフトウェアプロセス知識の獲得-", 人工知能学会誌, Vol.11 No.4, pp.97-103, 1996.
20. 落水, 門脇, 藤枝, 堀:「ソフトウェア分散開発支援環境「自在」のアーキテクチャ設計」, 電子情

報通信学会 ソフトウェアサイエンス研究会資料, SS94-18, 1994.

21. 落水:「ソフトウェア・レポジトリ」, 情報処理,Vol.35, NO.2, pp.140-149, 1994.

22. J.Conklin and M.Begeman:"gIBIS: A Hypertext Tool for Exploratory Policy Discussin", CSCW'88, pp.140-152, 1988.

23. C.Kadowaki, K.Ochimizu: "Recording the Progress and the Reasoning of Deliberations",(submitted to IPSJ).

24. Gio Wiederhold: "Mediation in Information Systems", ACM Computing Surveys, Vol.27, No.2, pp.265-267,June 1995.

25. Sandra Heiler: "Semantic Interoperability", ACM Computing Surveys, Vol.27, No.2, pp.271-273,June 1995.

26. Frank Manola: "Interoperability issues in Large-Scale Distributed Object Systems", ACM Computing Surveys, Vol.27, No.2, pp.268-270,June 1995.

27. D.Pascot 著、落水訳: 「C/S データベース設計入門」, 日経ＢＰ社,1997.

28. 堀, 落水:「ソフトウェア開発における自己反映オブジェクト指向モデルに基づく共有情報の管理法」, コンピュータ・ソフトウェア,Vol.13, NO.1, pp.37-54,1996.

29. M.Hori, Y.Shinoda, K,Ochimizu: "Shared Data Management Mechanism for Distributed Software Development Based on a Reflective Object-Oriented Model", LNCS 1080, Advanced Information Systems Engineering, pp.362-382, 1996.

30. D.Mandrioli, A.Morzenti, P.San Pietro, E.Crivelli: "Specification and verification of real-time systems in a logic framework: the TRIO environment", , Vol., No., pp.-,.

31. S.Bandinelli, A.Fuggetta, C.Ghezzi: "Software Process Model Evolution in the SPADE Environment",IEEE Trans. on SE, Vol.19 NO.12, pp.1128-1144, Dec.1993.

32. 門脇, 落水:「ソフトウェア・プロセス実行における系統的コミュニケーション支援の一方式」, Jaist Research Report, IS-RR-97-0009S, 1997.

33. 近野, 門脇, 落水:「グループウェアベース「栞」を用いた電子会議内容の進捗把握と文書化の支援」, 情報処理学会ソフトウェア工学研究会資料、107-12,1996.

34. 川上, 川瀬, 小寺, 鈴木, 萩原, 橋本, 落水:「ネットワークを介した協調活動の支援環境」, Jaist Research Report, IS-RR-96-0012S, 1996.

35. D.Jennings:"On the Definition and Desirability of Autonomous User Agents in CSCW", CSCW and Artificial Intelligence, Springer Verlag, 1994.

36. H.Murakoshi, H.Kaiya, K.Ochimizu: "An Analysis of Obstruction in Cooperative Work over a Computer Network", Proc. of CICS'96,pp.265-267,June 1995.

37. 荒木、岡村、佐伯、三浦、落水、篠田、海谷:「ネットワークを介した協調活動の実現にむけて」, 情報処理学会、サマー・ワークショップ・イン・立山報告集, pp.105-112, 1995.