Supporting Dynamic Process Specifications using Communication based Processes

Inoue, Sozo
Graduate School of Information Science and Electrical Engineering, Kyushu University

Iwaihara, Mizuho
Graduate School of Informatics, Kyoto University

出版情報：Proc. 35th Hawaii Int'l Conf. System Sciences, 2002-01. IEEE Computer Society
バージョン：
権利関係：

Supporting Dynamic Process Specifications using Communication based Processes

Sozo Inoue
Graduate School of Information Science
and Electrical Engineering
Kyushu University
6–1 Kasuga-Koen, Kasuga-Shi,
Fukuoka 816-8580 JAPAN
sozo@c.csce.kyushu-u.ac.jp

Mizuho Iwaihara
Graduate School of Informatics,
Kyoto University
Yoshida-Honmachi, Sakyo-ku,
Kyoto 606-8501, JAPAN
iwaihara@i.kyoto-u.ac.jp

Abstract

In this paper, we introduce M-Trans system, which has an ability of recording the specification of the design process for the communicative process, which is designed in a discussion and is be specified incompletely, of creating the communicative process according to the specification, and of supporting dynamic process specifications utilizing the record of executed communicative process. The system is based on the model that provides integrated specification of a process and communication. The execution of the communicative process may be performed on parallel with the design process, and the design process and the implementation may have interactions for coordination. Moreover, we present a method for verifying consistency between the communicative process and its specification. Managing communicative process is important since it realizes adaptation to unexpected situations, including exception handling and dynamic re-composition of a process in WfMS.

1 Introduction

In the real world, there still exist collaborative processes which are out of control for Workflow Management Systems (WfMSs). We consider a communicative process, which is designed in a discussion and may not be specified completely. The execution of the communicative process may be performed on parallel with the design process, and the design process and the implementation may have interactions for coordination.

Managing communicative process is important since it realizes adaptation to unexpected situations, including exception handling and dynamic re-composition of a process in WfMS. Flexible process management is a critical requirement for WfMS because the real world is inevitably surrounded by frequently changing environment and/or unexpected human behavior.

M-Trans system(Message Transaction System) is a system which treats the design process as a same model as communicative processes, and supporting communicative processes and its design processes by capturing the communication among the participants. In this paper, we focus on interaction between design processes and the implemented communicative processes, and especially on the behavior that the process once executed locally or ad hoc is adopted repeatedly. We support this behavior by suggesting the associable designs in the record of the processes to the designers who participates in the design process. The system is based on the model[9, 11, 10] which provides integrated specification of a process and communication.

The rest of the paper is organized as follows: Section 2 introduces a communicative process and its design process. Section 3 describes the design model of M-Trans system, Section4 describes the dynamic specification using the record of processes, and Section 5 describes the experience on the real e-mail archives. After discussing related work in Section 6, we conclude the paper in Section7.

2 Communicative processes

In this section, we reveal the features of a communicative process, and address technical challenges.

2.1 Requirements for communicative processes

A *communicative process* is a process that is defined in a discussion. For the sake of convenience, we call the discussion for designing a process a design process, and we call the execution of a designed process an *implementation*

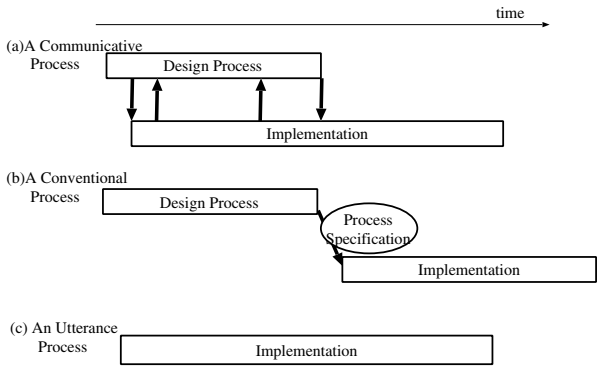


Figure 1. A classification of a process design

of the process. A communicative process is illustrated in Figure 1(a).

Traditionally, WfMS[4] is a system for managing processes that is previously defined by a designer. We call such a process a *conventional process* (illustrated in Figure 1(b)). The design process outputs the specification of the conventional process, and the implementation is performed after the completion of the design process.

In the literature, another type of processes, which is implemented by an utterance of a participant, are considered in [6, 13, 14, 1]. Basic human utterance, such as promises, orders, and questions are the targets of the system support. We call such a process which is created and proceeds by an utterance of a participant an *utterance process*(illustrated in Figure 1(c)). The process does not have a separated design process. The process is implemented by utterances from the participants in an ad hoc manner.

To clarify the features of communicative processes, let us consider an example process for publishing a program for a conference collaboratively. In the process, the program committee requests titles and authors' information to each author based on the predefined process. In the gathering process, the procedure and the method to gather the title varies according to each author. Some author may send the title by e-mail, and another may send it by fax. Some authors may want to send the title and authors' information separately. Some authors may need exceptional process to correct the title, such as requesting correction to the author. These details of the gathering processes are specialized with a negotiation and an agreement among the program committee and each author. After gathering the titles, the program committee edits the program, and assigns a chairman to each session.

The features of communicative processes are the following:

- A process is designed in the discussion. The process adopted in the discussion is implemented. In the example above, the process for gathering titles is constructed by an agreement between the program committee and each author.
- The implementation may even precede the design process. Since the design process is dominated by a discussion, there is not possibly enough time to specify the process or the domain knowledge completely. For example, the mistake of a title must be rapidly corrected to avoid the delay of the following tasks. In such a case, an implementation of a process without a formal specification might be performed.
- The design process is often rather dependent on its implementation than autocratic. The design process might continue after the process has been implemented. Then the major topic in the design process is the correctness of the implemented process, and deciding its effect on other processes. For example, after the program committee requires the author to resend the authors' information by fax as an exception because of a failure in printing, the exception is agreed in the design process, and discussed whether the exception is adopted as a normal option of the title gathering process. If it is adopted, the option sending by fax is incorporated into the process specification.
- A process is intermixed with utterance processes and conventional processes. In the example above, an agreement among the program committee and each author is performed by an utterance process. Moreover, the overall procedure for publishing a conference program can be specified, as a process which is readable for a WfMS, such as in order of gathering titles, program edit, and chairman assignment.

2.2 Flexible relations between a design process and the implementation

From the features of communicative processes shown above, we can figure out that the flexible management of relations between the design process and the implementation is required. Although, traditional WfMSs only support top-down approach, that is, the process is implemented after the design process is finished. We believe that the implementation-oriented approach for managing communicative processes is equivalently important.

We capture the constructions of relations between the specification and the instance as follows:

- (Consistency Check) First of all, the method for checking consistency between a design process and its implementation is necessary. This method would be performed frequently, because the consistency might be

violated not only when an instance is created, but also the design process is progressed.

- (Localization) The most simple, and powerful way to avoid the inconsistency between a design process and its implementation is to isolate them and to assume the cause of the inconsistency as a local adaptation. For example, to resend the authors' information is treated as a local adaptation for a particular author. Figure 2(b) illustrates the implementation without corresponding part in the design process by localizing the implementation of Figure 2(a).

- (Generalization) Localization is occasionally dangerous because the localized process can not follow up the progression of other processes, such that another process adopts to resend the authors' information, and the resending process need a confirmation steps by the program committee. Hence, we introduce design processes a method for incorporating the localized implementation and maintain the consistency. Figure 2(c) illustrates the generalization of a localized implementation and applying it to another process implementation.

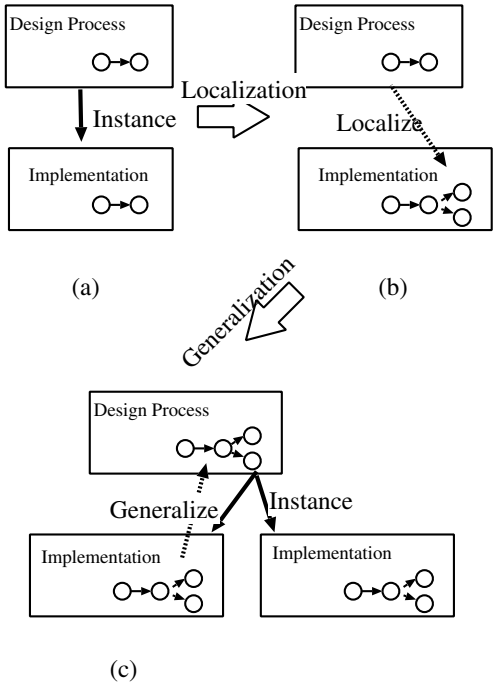


Figure 2. Localization and Generalization of a communicative process

Managing communicative process is important since it realizes adaptation to unexpected situations, including exception handling[2, 8] and dynamic re-composition[18, 20] of a process in WfMS. Flexible process management is a critical requirement for WfMS because the real world is inevitably surrounded by frequently changing environment and/or unexpected human behavior. In deed, adaptation to an unexpected situation is a communicative process, that is, the approach to be adopted is discussed in a communicative process, and the adaptation might be executed urgently on parallel with the discussion.

2.3 Design processes for communicative processes

A design process for a communicative process is an utterance process, because the design process proceeds with utterances of the participants. Many systems for supporting utterance process have been proposed. These systems [6, 13, 14, 1] classify utterances as propositions such as "suggest", "promise", and "cancel". Other systems aim to record the semantics of the discussion[5], in which each utterance is assumed to represent the state of the discussion. Moreover, goal-oriented decision rationale model[17, 19] is based on the vocabularies that divide (such as "subgoal"), solve(such as "achieve"), and evaluate(such as "support", "deny") problems and requirements. [16] introduces the specific vocabularies which specify the process design, such as "Has-Subtask", "Has-Temporal-Relationship", and "Has-Attribute".

From these approaches for supporting utterance process, we observe that the following properties, which are useful for implemented communicative processes, can be obtained from the history of the design process:

- The problem to be solved in the communicative process is divided into subgoals by the discussion. For example, to publish the conference programs is the problem. The problem may be divided into the subgoal to gather the titles and the subgoal to edit the program.

- Some utterances specify the process structure directly, as in [16].

- An utterance has possibly multiple responses. For example, a question may be answered by two or more answers, and any number of authors may submit the titles in the title gathering process.

We basically adopt such approaches. Figure 3 illustrates the architecture of processes in our model. The system specifies rules for designing a communicative process in a *design process template*, which is specified as an mg-template(described in Section 3.3), and the design process progresses along with the template. The record of the design process is viewed as a *process template*, which is also specified as an mg-template. Converting a design process

and process template is beyond the interest of this paper. A communicative process is implemented by instantiating the process template.

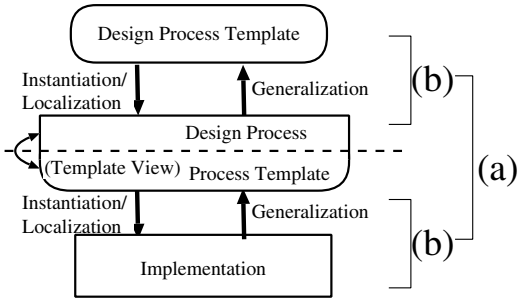


Figure 3. The process architecture in M-Trans system

Additionally, we adopt the generalization mechanism to realize practical support for design processes. We believe that not only the top-down approach based on the predefined rules but also the bottom-up approach that utilizes the already implemented processes is important. In practical situations, the process might not be described completely, because the process might be decided without a discussion, or the process might require a rapid execution. As shown in Figure 3, generalization and localization can occur both in the relation between the design process template and the design process, and between the process template and the implementation. Generalization mechanism is useful to explore the process structure effectively, because the generalization can be performed only for the repeated use of the process structure.

3 M-Trans system

In this section, we describe the M-Trans system.

The current M-Trans system is implemented as a Java servlet and Java applet. Figure 4 is a main user interface of M-Trans system. Users can access the system through web browsers. The outputs of the system are written in XML (eXtensible Markup Language). This schema provides an easy customization of the outputs from the system according to the application domain or the project.

We briefly overview the system in Section 3.1, describe m-group, which is the model for communicative processes in Section 3.2 and mg-template, which is the model for the m-group specification in Section 3.3, and address the consistency between an m-group and an mg-template in Section 3.4. We do not distinguish the design processes and other communicative processes (we ignore (a) in Figure 3),

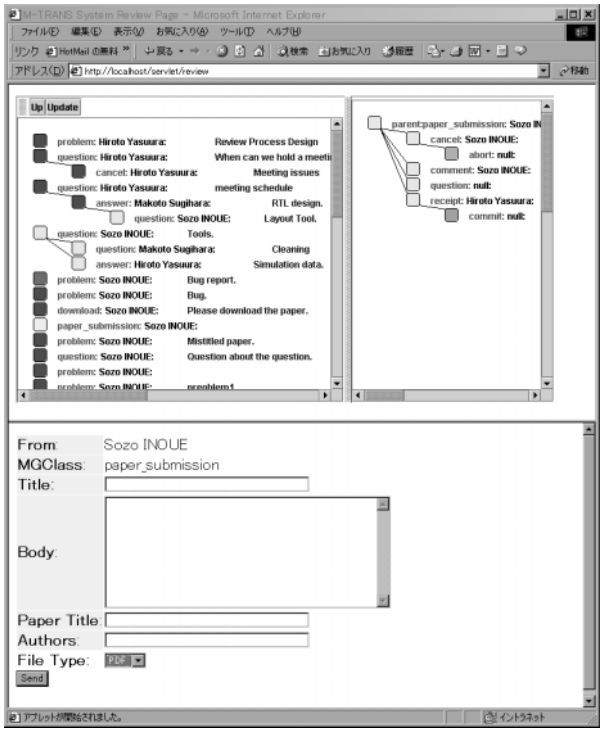


Figure 4. Main User Interface of M-Trans system

and only focus on the relation between the template and instance in this section (we focus on (b) in Figure 3).

3.1 The system overview

Design processes for a communicative process and a conventional process, and implementations of an utterance process and communicative process are mainly performed by communications. M-Trans system integrates workflow management and asynchronous communication among participants, by m-group data model, which is described in Section 3.2. The system has the following features:

- The system can execute an utterance process and construct a communicative process.
- The system can execute a communicative process without defining the full specification of the conventional process. This is realized by the localization mechanism. Users can start a process in an ad hoc manner, or the design process can progress without a restriction by the design process template.
- A method for verifying consistency between implementation and the specification is provided.

- The system can incorporate a localized implementation into its template by a generalization. Users can assure the implemented process in an ad hoc manner so that it is applied to another implementation.

The execution of an utterance process is realized by preparing the specification of human utterances and executing the utterance process by implementing the specification. The construction of a communicative process is realized by preparing the specification of the design process and running the design process by instantiating the specification, or binding the specification afterwards. The specification of design processes is introduced from decision rationale models such as [17, 16, 19]. Both specifications are given in the form of mg-template, which we describe in Section 3.3. Since an mg-template can also specify the conventional process, it is easy to evolve a communicative process to a conventional process without lacking the context of the communicative process.

3.2 m-groups

To integrate communication and processes, we define m-groups, whose example is shown in Figure 5(b). An *m-group* is,

$$[m\_id, m\_class, Children, Tdeps, body, Rscs, Roles].$$

m_id is an ID. $body$ is a participant's message by any medium such as text, voice, or video. m_class is an *mg-class*, which is an utterer's intention. Some examples of mg-class are "Issue", "Argument", and "Position" in gIBIS[5], and other examples are "Bug_report", "Request_modify", "Modify_design", and "Test_report" in software process [15]. An m-group is able to specify a semantically continuous human interaction with $Children$, called *child group*, which is a set of m-groups. An m-group which is allowed to have child groups is called *multiple m-group*, and the other m-groups are called *single m-groups*. An m-group m produces an event *root*, *execution*, *commit*, or *abort* each of which denotes a creation of m, a creation of a child group, a successful completion of m's intention, or a failure of the intention, respectively. An m-group can commit after its all child groups have *terminated*, that is, committed or aborted. $Roles$ is a set of *role*s, each of which is a participant to the m-group, and $Rscs$ is a set of *resource*s, which are a set of relative information, such as data used in the m-group.

$Tdeps$ is a set of *transactional dependencies (t-deps)*, which are represented as arrows between m-groups in the figure. We represent an t-deps from an m-group m_1 to m_2 as $(m_1 \rightarrow m2)$.When a t-dep is assigned between m-groups either of which has not terminated, the t-dep means that, equivalent to the control flow among activities in WfMS, the destination m-group cannot be executed before the source m-group commits. On the other hand, a t-dep means that the destination m-group refers the source m-group when the t-dep is assigned between m-groups that have been terminated. The latter t-dep corresponds to the reply to the previous argument. A t-dep has a label which is an element of the predefined set of strings. In the figure, root, commit, and abort events are shown as double circles.

3.3 mg-templates

For the purpose of describing a specification for an m-group, we introduce *m-group templates (mg-templates)*. An mg-template is,

$$[t\_id, t\_class, TChild, TDTmpls, TRscs, TRoles].$$

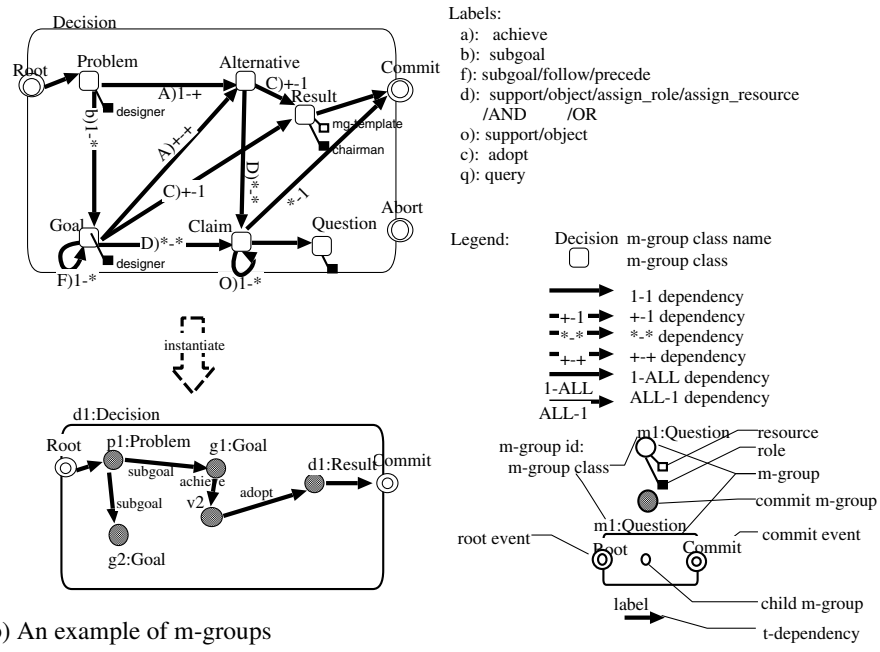
t_id is an ID, t_class is an mg-class, $TChild$ is a set of mg-classes, $TDTmpls$ is a set of tdep-template(described below in this Section), $TRscs$ is a set of resources, and $TRoles$ is a set of roles. mg-template is a description of a specification for an m-group with a specific mg-class. We represent an mg-template with an mg-class t_class as *mgTmplate*(t_class). Figure 5(a) is an mg-template for mg-class `Decision`, the structure of which is introduced from SIBYL[17].

While the definition of mg-templates seems to be similar to that of m-group, the following differences exist: an mg-template has a set of mg-classes instead of child groups, and an mg-template has a set of *tdep-templates* instead of t-deps.

A *t-dep template (tdep-template)*, drawn as an arrow in Figure 5(a) , denotes a t-dep when the mg-template is *instantiated*, which is the creation of an m-group according to the mg-template definition. We represent a tdep-template from an m-group m_1 to m_2 as $label(m_1[multi_1 \rightarrow multi_2]m_2)$, where $multi_1$ and $multi_2$ are *multiplicity constraints*, described below in this Section. A tdep-template connects an mg-class, a role, or a resource to one of them. Split or join properties in WfMS[4], such as AND-split or OR-join for t-deps, can be assigned for tdep-templates which share an mg-class, if the both ends of each tdep-templates are mg-classes. AND-split, or OR-split, means that either all or none of the destination m-group, or only one of the destination m-groups, is instantiated, respectively. AND-join, or OR-join, means that the instantiation of the destination m-group is allowed after all of the source mg-classes are instantiated, or one of the source mg-classes is instantiated, respectively.

If each ends of a tdep-template are mg-class, it has constraint, called *multiplicity constraint*, on the multiplicity of the instances at the time the instances commit. The constraint is 1, *, or + . The semantics of each constraint is the following: With an instance of the opposite end m-group,

(a)An example of mg-templates



(b) An example of m-groups

Figure 5. An example of m-groups and mg-templates

- 1: one instance of the m-group on the close end is connected by an instance of the tdep-template.
- *: any number of instances of the m-group on the close end are connected by instance of the t-dep.
- +: more than zero instances of the m-group on the close end are connected by instances of the t-dep.

We formally define multiplicity constraint.

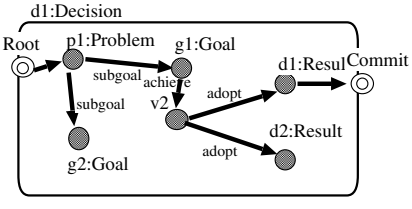
Definition 1 *(multiplicity constraint)*
A tdep-template t: $tlabel(t\_class_0[multi_0 \rightarrow multi_1]t\_class_1)$, is said to be *legal* w.r.t. an committed m-group [m_id, m_class, $Children$, $Tdeps$, $body$, $Rscs$, $Roles$] iff the following conditions satisfy:

- For every committed m-group $m_j \in Children$ whose mg-class is t_class_1, if $multi_0$ is,
  - 1: one t-dep $(m_{i_j} \rightarrow m_j)$ is in $Tdeps$,
  - +: more than zero t-deps $(m_{i_j} \rightarrow m_j)$ are in $Tdeps$,

  where, in both cases, each m_{i_j} is an m-group with mg-class t_class_0 in $Children$.
- For every committed m-group $m_i \in Children$ whose mg-class is t_class_0, if $multi_1$ is,
  - 1: one t-dep $(m_i \rightarrow m_{j_i})$ is in $Tdeps$,
  - +: more than zero t-deps $(m_i \rightarrow m_{j_i})$ are in $Tdeps$,

  where, in both cases, each m_{j_i} is an m-group with mg-class t_class_1 in $Children$. □

3.4 Consistency between an m-group and an mg-template



illegal w.r.t. "achieve(Goal[+ ->+]Alternative)" and "adopt(Alternatie[+->1]Result)"

Figure 6. An illegal m-group

For a tdep-template a t-dep is created on the instance of mg-template when the destination mg-class is instantiated. Moreover, the instantiation of the source mg-classes precedes the instantiation of the destination mg-class.

However, communicative process might be implemented against its specification as addressed in Section 2.1. For this reason, we define a legalexecution of an m-group as a consistency between an m-group and an mg-template.

Figure 6 is an example of illegal m-group w.r.t. tdep-templates in Figure 5(a). The m-group in the figure is illegal w.r.t. achieve(Goal[+ → +]Alternative), since $g2$ has no t-dep connected to Alternative. In the real situation, this corresponds to the case where not all the subgoals are considered with alternatives. Moreover, the m-group is illegal w.r.t. adopt(Alternative[+ → 1]Result) since two Results connected to $v2$ have been created, which means two results are adopted in the design process.

We define the legal execution of an m-group.

Definition 2 An m-group m: [m_id, m_class, $Children$, $Tdeps$, $body$, $Rscs$, $Roles$] is said to be *legal* iff the following conditions satisfy, where [t_id,t_class, $TChild$, $TDTmpls$, $TRscs$, $TRoles$] is $mgTmplate(m\_class)$:

- For every $child \in Children$, the mg-class of $child$ is in $TChild$.
- Every $tdtmpl \in TDTmpls$ is legal w.r.t. m if m has committed.
- For every tdep-template $tlabel(t\_class_1[multi_1 \rightarrow multi_2]t\_class_2) \in TDTmpls$, and every m_1 with t_class_1 and m_2 with t_class_2, where $m_1, m_2 \in Children$, a t-dep $(m_1 \rightarrow m_2)$ exists in $Tdeps$, where $tlabel$, $multi_1$, $multi_2$ are any values.
- Every $rsc \in Rscs$ is in $TRscs$.
- Every $role \in Roles$ is in $TRoles$.
- Every $child \in Children$ is legal. □

This definition constrains the mg-class, multiplicity constraint, resources, roles, and recursive legality of child groups of an m-group.

We present an algorithm **match** for checking if an m-group is legal, and obtains the components which causes the illegality. In the algorithm, the subroutine that checks if an m-group m satisfies the multiplicity constraint of a tdep-template t in m is specified as **checkMultiplicity**(t,m) (We omit its description).

Input: m-group m: $[m\_id, m\_class, Children, Tdeps, body, Rscs, Roles]$,
mg-template: $[t\_id, t\_class, TChild, TDTmpls, TRscs, TRoles]$

Output: a set of illegal components.

Algorithm: match

1. Let ret be an empty set.
2. If ($m\_class \neq t\_class$), add m to ret and return ret.
3. For each $child \in Children$,
   (a) Let $class_{child}$ be the mg-class of $child$.
   (b) If ($class_{child} \notin TChild$), add $child$ to ret.
   (c) Let $tmpl_{child}$ be $mgTmplate(class_{child})$.
   (d) If $\mathbf{match}(child, tmpl_{child})$ is not empty, add the obtained return values to ret.
4. For each t-dep $(m_1 \rightarrow m_2) \in Tdeps$,
   (a) Let $m\_class_1(m\_class_2)$ be the mg-class of $m_1(m_2)$, respectively.
   (b) If (a tdep-template $(m\_class_1[multi_1 \rightarrow multi_2]m\_class_2) \notin TDTmpls$), where $multi_1$ and $multi_2$ are any multiplicity constraint, then add $(m_1 \rightarrow m_2)$ to ret.
5. For each $tdtmpl \in TDTmpls$, add m to ret if ($\mathbf{checkMultiplicity}(tdep, m) = false$).
6. For each $rsc \in Rscs$, add rsc to ret if ($rsc \notin TRscs$).
7. For each $role \in Roles$, add $role$ to ret if ($role \notin TRoles$).
8. return ret.

4 Supporting dynamic specifications

In this section, we describe the flexible management of relation between an m-group and an mg-template. As discussed in Section 2.2, we assume that the localization and the generalization of a process specification, (mg-template in our model) are used in the following manner:

- For a rapid or an exceptional processes, localization of the mg-template is performed. This must be possibly performed without any discussion in the design process to be quickly applied to the urged processes.
- Localized m-groups can be generalized and reused if it is agreed in the design process. The generalized m-group is incorporated into its mg-template, and another m-group can be instantiated using the mg-template.

The system captures an *operation sequence*, which is a creation of an m-group, a commit or abort of an m-group, or a modification on mg-template which is a template view

of the design process. For an operation sequence, the system checks if the operation sequence is *safe*, that is, the existing mg-templates and their m-groups is legal even after performing the operation sequence. If it is safe, the operation sequence can be performed immediately. Otherwise, the system shows the cause of the unsafety to the users who request the operation sequence, and let them select among the following:

- Cancel the operation sequence and do not perform it.
- Perform the operation sequence locally, and do not affect the mg-template of the target m-group by localizing the m-group.
- If instance of the mg-template of the target m-group has a local structure which can apply to the result of the operation sequence, the local structure is generalized to the mg-template.

In the following, we describe the method to check if the operation sequence is safe, to detect the cause of the unsafety and show them to the users, to localize the operation sequence, and generalize a local structure of an instance to its mg-template.

Detecting the cause of the unsafety

To check if the operation sequence is safe, the system applies **match** algorithm. This procedure collects the m-groups which causes the unsafety. Let x the target component, which is an m-group, a resource, a role, or a t-dep, in an element, such as "add an m-group m;", in the operation sequence, let m_x be the m-group which includes x, and let T_p the mg-template of which m_x is an instance. If **match**(m_x, T_p) returns empty for any x in the operation sequence, then the operation sequence is safe, or else, the return value of the algorithm **match** is treated in the following procedure.

Visualizing the unsafety

If the operation sequence is safe, the operation sequence is performed immediately. If not, the system shows the causes graphically to the users. Figure 7 is the window which shows the causes. In the left frame in the figure, the part which is modified by the operation sequence, which is resending a paper by adding new m-group in this figure, is emphasized with a distinguished color.

This window can be monitored by a participant of the m_x. On the window, the users are able to select "Cancel", "Localize", or "Modify Template". "Cancel" denotes that the system cancels the operation sequence. In this case, the system clears the buffer which stores the operation sequence. We describe the remaining two selections in the following.

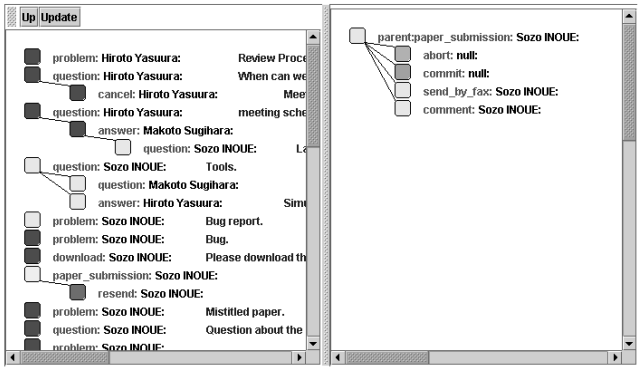


Figure 7. Visualizing the unsafety and candidates for generalization

Localization

If the users select "Localize", the system perform the operation sequence as a local effect in m_x. In this case, the localized m_x is ignored in the later **match**.

Generalize the local structure

If the user select "Modify Template" and if the result of the operation sequence is also included in some existing local instance, the mg-template is allowed to merge the local structure with itself. In this case, the local structure is called to be generalized.

We describe the method for generalizing a component in an instance to the mg-template T. We assume x' an m-group, a resource, a role, or a t-dep. The procedure for generalizing x' to T is the following:

1. If x' is a resource or a role, Add x to T.
2. If x' is an m-group,
   (a) Add a new mg-class c which corresponds to x' to $TChild$ in T,
   (b) create a new mg-template T_c which corresponds to the mg-class c, and,
   (c) generalize each child group to the created mg-template T_c by applying this procedure recursively.
3. If x' is a t-dep, add tdep-template "$c_s(1 \rightarrow 1)c_d$" to T, where $c_s(c_d)$ is the mg-class of the source (destination) m-group of x', respectively.

If the generalized mg-template matches x' and all the instances of T, T is allowed to be replaced with the generalized mg-template.

The system seeks the candidates to be replaced, and shows the users the difference from the original mg-template. The right frame in Figure 7 is the visualization of the candidates. In the figure, the m-groups which can be generalized are listed.

When the user selects one of the listed mg-templates, the system replaces T with the selected generalization. Finally, the operation sequence is performed. Figure 8 shows an example of generalization.

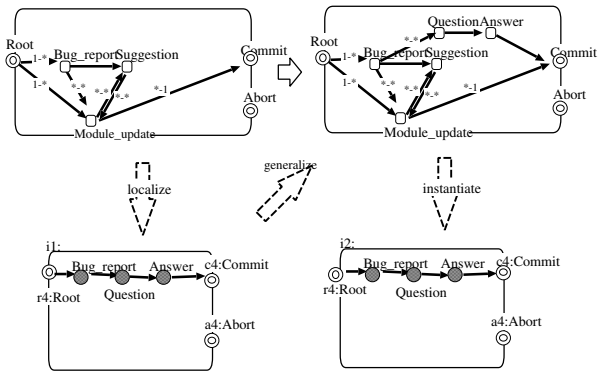


Figure 8. An example of generalization

5 Experience

We applied the e-mail archives in a software development project to our model manually, and analyzed the effectiveness. 9 developers and about 15 test users participated in the project. We analyzed about 130 e-mails which are exchanged among the participants during about 2 weeks.

We attached an mg-class to each e-mail by considering the context of the e-mail, and we assumed each e-mail is an m-group. To see the effectiveness of localization and generalization, we prepared only an empty mg-template, and constructed an mg-template using localization and generalization technique. We assumed that a reference relation by a replying e-mail is a t-dep. We counted the number of e-mails which caused an instantiation, localization, or generalization of an m-group.

We show the result in Figure 9. The figure is the transition of the number of exchanged e-mails. The e-mails are classified into the effects on an m-group and an mg-template by every 2 days. From the figure, we can observe that the localization shares a large part. This demonstrates the effectiveness of localization, because there exists a lot of m-groups that is executed locally without its mg-template. The rate of generalization is small, and the rate of instantiation, which could occur as a result of a generalization, is much larger than the rate of generalization. This suggests that a particular rate of m-groups can be managed in a same manner as conventional processes by utilizing generalization, even if the initial mg-template is empty. Moreover, the rate of instantiation tends to slightly increase. Hence, more m-groups would be managed as a consistent mg-templates instance. Figure 8 is one case of the generalization which found in the analysis.

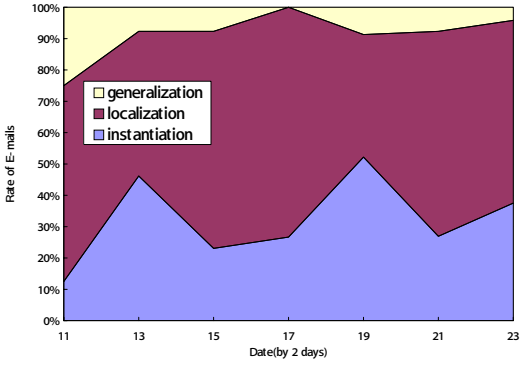


Figure 9. The rate of emails in each category

6 Related work

One of the requests remained for supporting collaborative work is adaptation to unexpected situations. The real world is inevitably surrounded by frequently changing environment and/or unexpected human behavior. As for process management technology, dynamic and adaptive workflow systems for managing exceptional situations have been proposed[12, 20]. A formalization of split/join methods is proposed for dynamic restructuring of transactions for tree-structured human activity in [18]. In [7], an adaptation to a critical exception by evolving partial specification of a process is realized. Moreover, flexible implementation of a process is introduced, which contains cloning an activity to multiple participants and an activity that is allowed to be implemented anytime in the lifetime of the process. However, these approaches focus on the flexibility in a process instance or flexible usage of 'strict' specifications, and do not consider utilization of the flexible specifications based on the semantics of human activity, such as policies for decision-making and negotiation among organizations. We introduced an approach which utilizes flexible specification, which may be in incomplete state, by capturing communication and providing verification method. [3] proposes an approach to promote reusability by extending other processes and to define rules for handling exceptions using ECA rules. Although ECA rule is effective for an expected exception, it cannot adapt to an unexpected behavior in a communicative process, since the ECA rule must be previously defined.

[12] proposes several flexible methods for re-binding instances to a new version of specification, such as eager propagation(the change of a specification immediately affects all its instances), and lazy propagation (new version is created). This approach would be also useful in communicative process, and can be combined to our approach. Our approach assumes re-binding as a significant method to construct a specification, and applies it to both communicative processes and its design processes.

Many decision rationale models have been proposed in the literature[5, 17]. These models aim to capture the design rationale, which consists of the design problems, alternative solutions, tradeoff analysis among the alternatives, and the decisions which had been made. The models are based on a semi-formal message of participants. The models are powerful and generic models, but do not have enough information to implement a communicative process, such as the order of execution, participants, resources, and dependencies among the implementations of communicative processes. iDCSS[16] introduces specific vocabularies to construct a process, and integrates decision rationale model and process management. iDCSS is based on similar motivation to our approach, however, it does not consider the flexible relation between a specification and a process instance.

7 Conclusion

M-Trans system provides the mg-templates for constructing the specification in a communication, and provides the functionality to dynamically modify the mg-templates by checking the consistency between its instances and using the record of the instances. M-Trans system has a wide applicability because it is able to specify and execute an utterance process, a communicative process, and a conventional process.

References

[1] D. P. Bogia, W. J. Tolone, S. M. Kaplan, and E. de la Tribouille. Supporting dynamic interdependencies among collaborative activities. In *Proc. ACM Conf. Organizational Computing Systems*, pages 108–118, 1993.

[2] F. Casati, S. Ceri, S. Parabosci, and G. Pozzi. Specification and implementation of exceptions in workflow management systems. In *ACM Trans. Database Systems*, volume 24 of *3*, pages 401–451, Sep 1999.

[3] D. K. W. Chiu, K. Karlapalem, and Q. Li. E-ADOME: A framework for enacting e-services. In *Proc. 1st Workshop on Technologies for E-Services*, Egypt, Sep 2000.

[4] T. W. M. Coalition. Terminology and glossary. In *Technical Report WFMC-TC-1011, The Workflow Management Coalition*, Jun 1996.

[5] J. Conklin and M. L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. In *ACM Trans. Office Information Systems*, volume 6 of *4*, pages 303–331, Oct. 1988.

[6] F. Flores, M. Graves, B. Hartfield, and T. Wingrad. Computer systems and the design of organizational interaction. In *ACM Trans. Office Information Systems*, volume 6 of *2*, pages 153–172, April 1988.

[7] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. Managing escalation of collaboration processes in crisis mitigation situations. In *Proc. 16th Int'l Conf. Data Engineering*, pages 45–56, 2000.

[8] E. E. in chief, V. Number, P. Hagen, and G. Alonso. Flexible exception handling in the opera process support system, 1998.

[9] S. Inoue and M. Iwaihara. Structured message management for group interaction. In *Proc. Int'l Workshop. New Database Technologies for CSCW and Spatio-Temporal Data Management (NewDB'98)*, Singapore, Nov. 1998.

[10] S. Inoue and M. Iwaihara. Adapting transactions to exceptional situations using structured messages. In *Proc. Int'l Symp.Database Applications in Non-Traditional Environments(DANTE'99)*, pages 264–271, Kyoto, Nov. 1999. IEEE Press.

[11] S. Inoue and M. Iwaihara. Dynamic composition of transactions on asynchronous communication. In *IPSJ Trans. Database*, number SIG 8 (TOD 4),Vol. 40, pages 1–12, Nov. 1999.

[12] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *Proc. of 3rd IFCIS Int'l Conference on Cooperative Information Systems(CoopIS '98)*, New York, Aug. 1998.

[13] S. M. Kaplan, A. M. Carrol, and K. J. MacGregor. Supporting collaborative process with conversationbuilder. In *Proc. ACM Conf. Organizational Computing Systems*, pages 69–79, 1991.

[14] S. M. Kaplan, W. J. Tolone, D. P. Bogia, and C. Bignoli. Flexible, active support for collaborative work with conversationbuilder. In *Proc. ACM Conf. Computer-supported Cooperative Work*, pages 378–385, November 1992.

[15] M. I. Kellner, P. H. Feiler, A. Finkelstein, T. Katayama, L. J. Osterweil, M. H. Penedo, and H. D. Rombach. Software process modeling example problem. In T. Katayama, editor, *Proc. 6th Int'l Software Process Workshop*, pages 19–29. IEEE Computer Society Press, 1990.

[16] M. Klein. iDCSS: Integrating workflow, conflict and rationale-based concurrent engineering coordination technologies. In *CERAs*, 3 1995.

[17] J. Lee. SIBYL: A tool for managing group decision rationale. In *Proc. Conf. Computer-supported collaborative work*, pages 79–92, Oct 1990.

[18] L. Liu and C. Pu. Methodological restructuring of complex workflow activities. In *Proc. 14th Int'l Conf. Data Engineering*, pages 342–350, California, 1998.

[19] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach, 1992.

[20] M. Reichert and P. Dadam. ADEPT$_{flex}$ – supporting dynamic changes of workflows without loosing control. *Journal of Intelligent Information Systems*, 1997.