

Generalized Process Structure Grammars (GPSG) for Flexible Representations of Work

Natalie S. Glance, Daniele S. Pagani, and Remo Pareschi

Rank Xerox Research Centre, Grenoble Laboratory
6, chemin de Maupertuis - 38240 Meylan - France
{glance, pagani, pareschi}@grenoble.rxc.xerox.com

ABSTRACT

The promise of workflow solutions for coordinating organizational processes is currently being obscured by strong criticism of the rigidity of their work representations. This rigidity arises in part from viewing work processes as unfolding along a single line of temporally chained activities. In reality, work evolves both horizontally, in the cooperation of causally unrelated, but information-sharing tasks, and vertically, in the coordination of causally-dependent activities. In this paper, we present our process modeling approach which (1) views documents and tasks as duals of each other, capturing horizontal cooperation; and (2) exploits *constraints* to express the *soft dependencies* among related activities and documents within the framework of generative rule-based grammars for processes, thus handling vertical coordination.

1. INTRODUCTION

Although workflow approaches to the coordination of activities in organizations have drawn much interest, they have also been criticised for the rigidity of their organizational process models. Strong evidence concerning the risks related to the hasty adoption of workflow technology comes from ethnographers and social scientists who study office work practices [20] and the impact of workflow support systems on everyday work [5, 18]. These studies show that when models of work processes are removed from their original context and embedded into systems that drive and control human behavior, then major disruptions occur that may hinder workers' ability to get things done, forcing them to invent workarounds that bypass the system.

As Ellis, Keddara, and Rozenberg point out [7], counterposed to workflow approaches for collaborative work are a number of CSCW systems, such as group editors and shared workspaces, which intentionally lack representations of the organizational context and goals. These systems do not attempt to coordinate workgroup members to accomplish a goal, but simply provide an environment for sharing common artifacts [17] through mechanisms such as replication (e.g., Lotus Notes) and event notification (e.g., GroupDesk [10]). Other systems acknowledge the need for basic coordination

mechanisms by providing locking and versioning of shared documents, based on check-in and check-out operations performed by users. These tools do not dictate behaviour, but simply coordinate concurrent access and provide status information about other users' activities ("document X is locked because user Y checked it out at 10:35 am"). A natural extension of these basic coordination mechanisms is the definition of routes attached to shared documents ("send me document X after Y has checked it in, and then send it to Z"). Abbott and Sarin [1] define this direction as "document-oriented workflow," in contrast to "activity-oriented workflow," where processes are modeled as sequences of activities and documents are attached to activities.

Taking the rejection of systems that embed *rigid* representations of work as a starting point, we must, however, acknowledge users' needs for technological support of coordinated efforts. This naturally raises the issue of how to design systems that embed *flexible* representations of work. The coordination mechanisms provided by the CSCW systems described seem to fit on a spectrum ranging from unstructured work facilitated by documents sharing and versioning, to semi-structured routes attached to documents, to structured activity-oriented processes. To date, however, no CSCW system or work representation formalism is capable of spanning the entire spectrum, giving workers full choice about when to specify process representations, to what level of detail, and to what extent coordination should be delegated to the support system.

In the special issue on the Communications of the ACM on Representations of Work, Bannon concludes the Commentary saying "I do not think that anyone here is arguing the need for representations of work *per se*, rather the argument is over what it is that you are doing when you build representations initially, and how they are to be used in subsequent stages of the design process [4]." In a similar vein, Suchman [19] points out that "problems arise when normative representations are either generated at a distance from the sites in which the work they represent goes on, or are taken away from those sites and used in place of the work itself."

With these motivations, we propose a framework for representing, simulating and enacting work that allows:

- the use of flexible work representations that help workers reason about work and (re)plan activities, instead of rigid ones that dictate a predefined procedure;

- the location, adaptation and modification by workers of the most appropriate sequence of tasks to get things done (including short cuts, exception handling, etc.) while respecting the constraints of the work process (deadlines, rules, resource bottlenecks, etc.).
- the capture and enactment of different work coordination mechanisms, e.g. document-oriented vs. activity-oriented.

The main ingredients of our framework are *grammatical rules* to generate representations of work and *feature constraints* [2] to represent complex work artifacts and augment the basic capabilities of context-free grammars. We claim that, taken together, these will provide the flexibility needed to meet the requirements given above. Since our framework generalizes the grammatical approach to work processes described in Pentland [14, 15], we refer to it as *Generalized Process Structure Grammars* (GPSG).¹ Here are some characteristics of the GPSG framework that enable flexible process representation:

- *Open process definitions.* New rules can be added and existing rules can be deleted to allow incremental process design via the accumulation of local case-by-case definitions of subprocesses. In contrast, traditional modeling approaches employ the overall work representation as a template that must be defined and modified as a whole.
- *Dual representations of documents and activities.* A rule can either define how an activity decomposes into a number of sub-activities or how a document decomposes into a number of sub-documents. In either case, constraints are used to specify dependencies between activities and document states and between document parts and their associated activities.
- *Process definitions freed from the fixed overall order enforced by traditional workflow systems.* Flexible, or soft, temporal dependencies can be realized in several ways: through casual dependencies between activities; through triggers based on document states; and via informational dependencies between document parts.

The paper is structured as follows: in the next section, we elaborate upon the background to our approach. In section 3, we describe the GPSG formalism, giving simple examples to demonstrate its expressiveness. A more extensive example representing a collaborative authoring [6, 14, 15] process is outlined in section 4. Finally, we conclude in section 5 and discuss our ongoing work developing a simulation environment prototype for GPSG.

2. BACKGROUND

At the heart of our methodology lies a process grammar approach based on rules, objects, features, and constraints. The idea behind process/action grammars [6, 14, 15] is that definitions of work processes can be

generated from a set of rules and a lexicon of process objects just as this sentence was generated in accordance with the rules of English grammar and an English dictionary (we hope). Compare this with other approaches, such as those based on Petri nets and their variants [8, 22], which require the designer to specify the entire process ahead of time, complete with alternatives and task assignments. At best some approaches allow limited flexibility, such as a number of workflow systems which use the concept of “roles” to attach responsibility for a task to any of a group of people, and Regatta [21], a research prototype now available as TeamWARE Flow, which allows sub-plans to be elaborated on the fly. However, using a generative grammatical approach, a given process instance can be incrementally singled out from the space of possible workflows defined by the rules as the process evolves.

In this paper, we propose to build on the concept of process grammars and take them further by enriching and augmenting the representation of tasks and documents and the types of dependencies among them that can be expressed. Key to our approach is the use of constraints to describe the complex soft dependencies of actual work practice. While in the vernacular, constraints may evoke limiting the possibilities for action, in the formal language of our approach, a constraint describes the *set* of possibilities allowed, which collapses onto one choice only when the time for action arrives. Here’s one way to think about the advantage of a constraint-based approach: other modeling approaches require the designer to cut out a process map (perhaps leaving some details to be elaborated later) from the complex fabric of collaborative work, discarding most of the rich weave. Using constraints, the designer can snip away at the background, allowing the outlines of the process to gradually emerge in the foreground during enactment.

The use of constraints to flexibly describe work processes has also been explored by Florijn et. al. [9] in their coordination language HOPLa, but in the context of *parsing* a given sequence of actions (*i.e.*, checking the process instance for “grammatical” correctness), as opposed to our approach, which focuses on *generating* the space of potential work actions towards a goal. Bowers and Churcher [6] illustrate an approach close to ours, but they consider only temporal constraints defining partial orders in the actions of a process. By contrast, we use also feature constraints to describe the fine structure of process activities and artifacts, and how they relate to each other. Another aspect of our approach is that feature objects can be organized in inheritance hierarchies in the object-oriented programming style. Thus, while in this paper we focus on the dynamic aspects of constraint solving for the incremental run-time refinement of process models, feature constraints can also be used for building object-oriented libraries of processes as in the Handbook of Organizational Processes [13].

Finally, our methodology promotes both activities and documents to the rank of first-class citizen, in contrast with current work process modeling and enactment

¹ GPSG is also the acronym of the analogous extension of basic context-free grammars to handle the complexity of human languages (Generalized Phrase Structure Grammar). See [11].

tools that are primarily either *activity-oriented* or *document-oriented* [1]. In our formalism, activities and documents can be seen as duals of each other. Along one dimension, processes can be partitioned into component concurrent activities that are persistent and long-lived. Along a second dimension, processes can be partially represented by the evolution of documents and sub-documents.² The complementary interaction of activities and documents allows for a richer representation of processes. Activities trigger the creation and modification of documents; document states enable new activities and re-awaken dormant ones.

In the latter part of this paper, we will test our formalism against document-centered collaborative processes, which present a host of challenges both for process modeling and for enactment. One major problem for an enactment engine is providing adequate document access control among multiple users. This involves providing versioning control, limiting access to authorized users, merging parallel version streams, and a number of other issues. The mechanics of these operations must be left to the particular technology implementation; however, its semantics must be provided by the modeling language. A second challenge is designing a methodology that has the expressiveness required to represent the wide range of coordination methods observed in collaborative document-centered work.

Coordination methods and document structure and control are highly interdependent; that is, the type of coordination strategies that are appropriate will depend on the structure of the document and the technology available for controlling access. This connection can be observed in Posner and Baecker's [16] categorization of collaborative writing projects along four dimensions: roles, activities, document control methods, and writing strategies. In particular, we can think of document-centered processes as belonging to either one of two structural categories: decomposable document processes and non-decomposable processes, although in general, the process will be best expressed as a dynamically evolving combination of the two. In the first case, the process tasks map one-to-one onto the independent sections of a decomposable document. An example of such a process might be the writing of a book in the case where it can be broken down into almost independent sections such as the introduction, chapters, and conclusion. As a result, the coordination of activities for decomposable document processes can be very straightforward: each activity is associated with a portion of the document. In the second case of non-decomposable processes, there will be a many-to-one mapping of documents to tasks and tasks to documents. An example is the brainstorming, authoring, editing, and approval of a proposal, in which all steps of the process act upon the same physical document. There are two distinct coordination solutions for processes that require the sharing of a non-decomposable document:

- (1) sequential activities, where each actor works on a locked version of the shared document; and
- (2) concurrent activities, where the work of actors upon a frozen version is later merged.

In section 4, we will use our approach to model a collaborative writing process to show how process grammars can flexibly express the range in control and coordination required for document-centered processes. But first we explain the GPSG formalism in section 3 below, which finishes with several simple examples of how soft temporal and structural dependencies can be expressed for work processes in general.

3. GENERALIZED PROCESS STRUCTURE GRAMMARS (GPSG)

Traditional workflow systems have a Process Description Language (PDL) that allows users to define process templates. The system then allows users to instantiate process templates and execute process instances. In these workflow systems, a process template is represented as a legal phrase in the system's PDL. A process instance is the instantiation of the template. The flexibility of the process representation to adapt to the actual work practice and to changing conditions can be provided at two levels:

- Through conditional statements in the process template. This, however, requires changes to be anticipated during the design of the process template, i.e., before process execution;
- By allowing users to change the process definition during execution. Such changes, however, are always expressed as deviations from the norm (the original process template). Furthermore, dynamic changes of process templates and process instances are a difficult problem, addressed by Ellis et al. [7].

The GPSG approach is different (see Table 1). When the user wants to define a process template, s/he constructs a process grammar which specifies the lexicon of process objects (e.g., activities, documents, roles) and the rules to combine them. A process instance is any legal phrase generated from the process grammar during simulation or enactment. Indeed, a process grammar template defines a *process space*, a potentially infinite number of process instances. As a result, the flexibility of work representations based on the GPSG formalism is much greater because the designer can specify the constraints of the process and the rules to define what is allowed and what is not allowed, rather than the exact sequence (and alternatives) of activities and events. The flexibility of the process specification emerges from the negation of the constraints (everything not excluded by the grammar rules is possible). In contrast, within PDL frameworks, a process instance is chosen by selecting among conditional branching statements. Furthermore, there is the possibility of allowing the user to modify the process grammar itself during execution; however, we will not elaborate on this question in this paper.

Applying a Cartesian metaphor to the concept of process space, we can think of the rules as defining the axes and the constraints as carving out the curve. There are two main categories of rules: *activity-centered rules*

² We are also exploring adding a third dimension that captures the importance of roles in collaborative processes through the use of role-based rules.

	Traditional workflow systems	GPSG approach
Workflow system	<i>PDL grammar</i> + <i>parser</i> of user-defined process + <i>interpreter</i> of process. <i>Lexicon</i> : activities, dependencies among activities.	<i>GPSG grammar</i> + <i>generator</i> of user-defined processes + <i>constraint solver</i> + <i>compiler</i> of process.
Process template	Legal phrase defined by the user respecting the PDL grammar.	User-defined process grammar; Lexicon: activities, documents; Dependencies, defined as feature constraints.
Process instance	Instantiation of process template.	Legal phrase generated by the user from the process grammar.
Flexibility in process instance	Conditional statements in process template. Change of process template.	New legal phrase in process grammar. New phrase in modified process grammar.

Table 1 Comparing traditional workflow modeling and Generalized Process Structure Grammars.

and *document-centered rules*. Activity-centered rules describe how goals break down into sub-goals and under what conditions. For example, the goal of getting to work in the morning could be broken down into getting up, washing, eating, and driving to work. Each of these goals could be further broken down using additional rules. Dependencies among the activities might be, for instance, that one only eats when hungry and only drives in bad weather. Likewise, document-centered rules describe how documents are decomposed into sub-documents. A local paper, for example, is composed of news stories, editorials, and columns, accompanied by appropriate photographs and illustrations. Structural dependencies might include adding an editorial and photograph to accompany a late-breaking story, but cutting out another article to maintain overall line limits.

Both activities and documents are *feature structures*, that is, objects described by a set of features, or attribute-value pairs, explained in more detail in section 3.2. The feature values may themselves be feature objects. Interdependencies between objects are described using *feature constraints*, explained in section 3.1 below. In section 3.3, we give a few simple examples of individual grammar rules to show how feature structures and constraints are combined with the rules to define dependencies among tasks and documents. Then, in section 4, we proceed to explain a process grammar segment for a collaborative authoring process, highlighting the benefits of our approach.

3.1 Constraints

Computational constraints are relationships between variables that have two properties: (1) they can be resolved at run-time rather than during the process definition; (2) they allow variables to take a range of values.

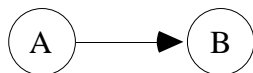


Fig. 1 Causal task dependency.

For example, in traditional workflow systems, the dependency between two activities A and B (Fig. 1) is usually expressed with the following variable assignment:

$$B.start := A.end \quad (1a)$$

where $B.start$ is the time when activity B starts, and $A.end$ is the time when activity A ends. The variable assignment is always computed in the same direction ($B.start$ is assigned the value of $A.end$). Using constraints, however, much greater flexibility can be achieved:

$$B.start \geq A.end \quad (1)$$

$$B.end < deadline \quad (2)$$

$$B.start \leq B.end - B.average_duration \quad (3)$$

The first constraint can be resolved from left to right or from right to left, depending on which variable is instantiated first. In a *task-pushed* execution mode, A is finished, $A.end$ is instantiated, its value is assigned to $B.start$, and then B starts. However note that, because of the \geq operator, B can start immediately or after some delay. Alternatively, if the execution mode is *goal-pulled*, $B.end$ may be instantiated because a deadline is looming (constraints 2 and 3): the value of $B.start$ is assigned to $A.end$ and A is executed.

Thus, by using constraints, we can define dependencies among tasks and documents more flexibly. Also, while project management systems allow temporal overlap of tasks, traditional workflow systems do not. Instead, these force either a temporally linear representation of tasks, where one task may not start until its precursor finishes, or a completely concurrent processing of tasks. With constraints, one may easily encode notions such as, “work on the paper may start any time after the literature search has begun.”

Likewise, instead of saying that writing the introduction to this paper must be handled by Daniele or Remo, say, we can instead prescribe that it may be handled by any of the authors, or perhaps, any author, but not Natalie, except for on Tuesdays. This kind of constraint also allows us to implicitly define who may accomplish the task by merely stating who *may not* be responsible for it, a capability not yet featured in any current workflow modeling or implementation systems.

Current workflow systems have a rigid notion of task assignment and timing, document routing and structure, which greatly limits the adaptability of the process definition to current situations and ad-hoc process

changes. Projecting their capabilities onto the concept of constraints, we could say that these use only limited constraints, such as equality (a task is done by a certain person, or one among a certain group of people, only at a certain time, or only after some other task has been completed). By adding other kinds of constraints, such as inequality (start task any time after 10 o'clock) and disjunction (task may be done by anyone who is not a manager), we can greatly increase the expressiveness of a process definition.

More generally, the GPSG approach uses an expanded set of constraints to flexibly specify when and under what conditions the process grammar rules are (or are not) fired. Constraints can control the timing of activities, the scheduling of resources (people, computers, machines), the structure of documents, and access control to documents, in a way that captures process variation as well as recognized routines.

3.2 Features

Features of objects, in combination with constraints, allow us to use rules to describe not only how tasks and documents break down into sub-tasks and sub-documents, but also how these sub-parts are interwoven into relatively loose or tight webs of interdependencies. For the most part, the semantics of features emerges from the local context of the rule. However, it is very useful to have a set of features whose definition remains constant across objects and rules. In particular, the *doc* feature of activity objects and the *task* feature of document objects are central to establishing the duality between task-based rules and document-based rules. These two related features establish an interconnection between the state transitions of a document and the state transitions of its dual task. Fig. 2 shows the allowed transitions between one activity state to another and between one document state to another and also (loosely) indicates how a state transition in one of a dual pair couples to a state transition in the other of the pair. (The interconnection becomes more complex if a document is paired with more than one task, or a task with more than one document.)

More specifically, the *doc* feature of an activity is a pointer to the document associated with the task. When the task is enabled, the document is created, if it does not already exist. Similarly, the *task* feature of a document is a pointer to the activity associated with the document. Upon creation of the document, the task becomes enabled, if it is not already enabled or active. Upon the release of a new version of the document, the activity is re-enabled, if it is not already enabled or active. An important side-effect of the duality between activity vs. document rules is that a collaborative process can be seen as being initiated either through the readying of a top-level task, or through the creation of a top-level document. Thus, the GPSG framework can model both activity-triggered processes and document-triggered processes, blurring the distinction between them. The example in section 4 in particular will highlight the duality of task-based and document-based rules.

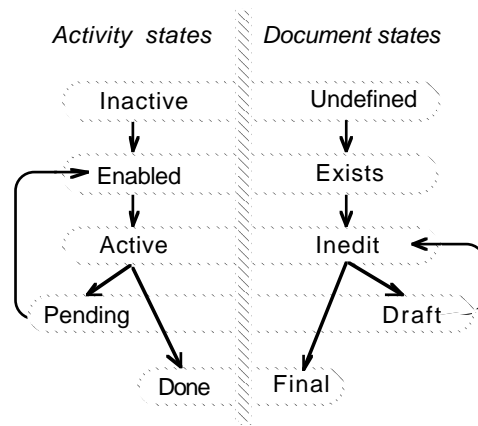


Fig. 2 Coupling of activity states with document states.

Both activity objects and document objects have an internally maintained *state* feature which points to a time record of changes in state of the object. Other pre-defined features of activities are: *role*, the group of users able to achieve the task; *duration*, the estimated duration of the task; *begin*, the start date of the task; and *end*, the end date of the task. Pre-defined features of documents include: *version*, a time record of incremental versions of the document; *group*, the persons authorized to create/modify the document; *length*, the length of the document (e.g., in bytes or pages); *parent*, a pointer to the parent document; and *contents*, a pointer to the physical contents of the document.

Other features of tasks and documents are defined implicitly within the local context of a rule to allow, in combination with constraints, hierarchical branching, iteration, and conditional rules, examples of which will be given in section 4 and to enforce deadlines, page limits, and role assignments, as we shall see in section 3.3 below.

3.3 Sample grammar rules

The sample grammar rules below are used to present ways in which basic, fundamental activity and document interdependencies may be represented using process grammar rules. In section 4, we will build upon and combine these representations to model a collaborative writing process that shows the power and flexibility of expression of the GPSG formalism.

Activity-centered rules: temporal dependencies

Here is an example of an activity-centered rule (variables local to the rule are identified by an initial capital letter) which we will use to show how to express (a) task concurrency; (b) task sequencing; (c) task overlapping; and (d) enforcement of deadlines.

```
|task    =goal| --> |task=task1| |task=task2|
::
|deadline=D |         |         |         |
|end       =E |         |         |         |

constraint = task1 precedes task2,
           constraint = E ≤ D.
```

The main body of the rule asserts that the task **goal** (the head of the rule) consists of doing **task1** and **task2** (the tail of the rule). The head must consist of exactly one goal, while the tail can contain arbitrarily many

subgoals. In addition, there is no implicit time ordering; *a priori*, **task1** and **task2** are concurrent. The features further describe the component tasks and can be used to pass values from one task to another, as well as from one rule to another, and to constrain the unfolding of tasks.

The symbol “::” separates the main body of the rule from (the arbitrarily many) constraints on the rule. The two constraints in the template limit how and under which conditions the rule may act. The first one constrains **task1** to finish before **task2** may start. Alternatively, we could replace this constraint with “task1 triggers task2,” which more laxly allows **task2** to start anytime after **task1** has started. The second constraint enforces a deadline, on **goal** directly, and on the component subtasks implicitly (by fiat, subtasks must begin after their parent task begins and end before their parent task ends).

Document-centered rule: structural dependencies

Next we have an example of a document-centered rule, which superficially appears very similar to an activity-centered rule because the semantic difference lies buried in its interpretation. The similarities emphasize the dual nature of task-based and document-based rules. Using this example, we show how to express (a) document decomposition; (b) document triggering of tasks; (c) document versioning; and (d) enforcement of page limits.

$$\left| \begin{array}{l} \text{doc} = \text{paper} \\ \text{length} = L \end{array} \right| \text{ --> } \left| \begin{array}{l} \text{doc} = \text{intro} \\ \text{task} = \text{wrIntro} \\ \text{length} = L1 \end{array} \right|$$

$$\left| \begin{array}{l} \text{doc} = \text{body} \\ \text{task} = \text{wrBody} \\ \text{length} = L2 \end{array} \right| \left| \begin{array}{l} \text{doc} = \text{concl} \\ \text{task} = \text{wrConcl} \\ \text{length} = L3 \end{array} \right| \text{ ::}$$

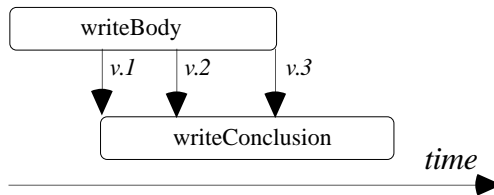
constraint = body precedes concl,
constraint = L1 + L2 + L3 ≤ L.

The main body of this rule asserts that the *paper* document decomposes into three parts: an *intro*, a *body*, and a *conclusion*. Once again, the head consists of exactly one document object and the tail of arbitrarily many sub-documents. However, while activity-centered rules focus on tasks and temporal dependencies, document-based rules focus on data and structural dependencies.

By default, all sub-documents can be accessed concurrently. However, in this example, the first constraint enforces that a final version of the *body* be complete before the *conclusion* can be created. The “precedes” constraint leads to the strict sequentialization of the two documents’ dual tasks: completion of the *body* (i.e., the dual task **wrBody** finishes) leads to creation of the *conclusion* sub-document (in turn enabling **wrConcl**, which may then become active at any time). The second constraint enforces a physical limit on the total length of the document.

Alternatively, if we would like to allow work on the body and the conclusion to proceed in parallel, but in a coordinated fashion, we could replace this constraint with “body triggers concl.” This constraint allows

work on the *conclusion* to start whenever work on the *body* has started, and also defines how the two efforts are coordinated: whenever a new version of the *body* is “released” by its “owners,” it becomes available to the owners of the *conclusion*:



In turn, this affects, **wrConcl**, the task dual to the conclusion sub-document: if the task was inactive or pending, it now becomes enabled. By this mechanism, changes in documents can lead to re-activation of long-lived activities. To our knowledge, no existing workflow system is able to represent long-lived tasks that re-awaken based on document states

4. EXAMPLE PROCESS GRAMMAR: MULTI-AUTHORING

To further illustrate the use of rules and feature constraints for modeling processes, we present a segment of a process grammar for the collaborative multi-authoring of a research paper (Fig. 3). (The full process grammar is given in [12].) The multi-authoring process illustrates the many coordination problems of a process involving multiple actors (with multiple roles) sharing a complex, structured document.

There are many possible alternative scenarios for the multi-authoring process. The scenario we have modeled describes the collaborative writing of a research paper, followed by submission to a referee and publishing, if accepted. The writing phase itself breaks down roughly into doing research, implementation, analysis, and writing. (Thus, the writing of a research paper is taken to be closely intertwined with the larger research agenda.) The research paper *qua* document is modeled as a complex structured document, roughly divisible into interdependent sections: it exhibits both decomposable and non-decomposable features. Some sections of the paper will have multiple authors and/or multiple dependencies on other sections and tasks. Over time, the paper becomes more or less decomposable with respect to the activities associated with it (see Fig. 4).

The principle roles are: author, consultant, copy-editor, referee, editor, and publisher. One actor can have multiple roles (potentially all of the above!). Authors write and research the paper; consultants advise from the sidelines. Copy-editors proof-read and comment. Referees review and judge papers. Editors coordinate reviews from individual referees. Publishers publish papers perceived as publishable.

Associated with the grammar rules are several global variables:

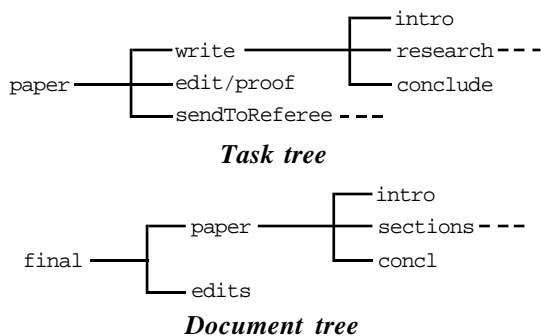
- the role sets: authors; consultant; copy-editor; referee; publisher.
- deadlines: deadlines for submission, refereeing, revising, and publishing.
- length: page limit for the paper

Given the space limit of this paper, this process description and its encoding is not intended to be complete, but instead to show that we can encode the complex coordination problems involved in a suitably flexible manner via process grammars.

Via the examples described in section 3.3, we showed how rules and constraints in combination can be used to express temporal and structural dependencies such as setting and propagating deadlines (from task to sub-task) and page limits (from document to sub-document). In the explanations that follow, we will illustrate how the process grammars can be used to model: (a) the decomposition of processes into subprocess and documents into sub-documents; (b) conditional branching among activities; (c) the interaction between activities and structured documents; and (d) activity coordination and version control for both decomposable and non-decomposable documents processes.

4.1 Hierarchical decomposition

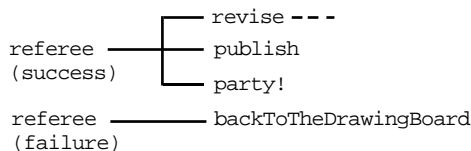
Sequences of rules are used to hierarchically break down the multi-authoring processes into tasks and subtasks and documents and subdocuments. The diagram below shows one possible task tree and its complementary document tree for the multi-authoring grammar:



The modularity of a rule-based approach makes it easy to redefine subprocesses and subdocuments. Replacing a rule defining a subprocess is like replacing a branch of the task or document tree. Adding a rule grows a new branch; removing a rule prunes a branch.

4.2 Conditional branching

Sets of rules can also be used to describe alternative subprocesses or subdocuments. The figure here shows two possible expansions of the **sendToReferee** subtask tree, depending on whether the referee accepts the paper or rejects it:



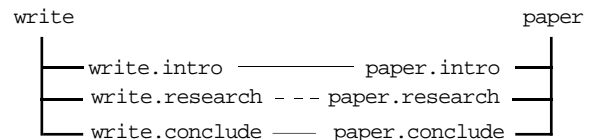
Using a rule-based approach, it is easy to add or remove alternatives by entering a new rule or deleting an old one. Also, it is important to note that while the example given here is purely deterministic, it is also possible to branch non-deterministically among several

possibilities. The evolution of the task and document trees would then have a random component.

4.3 Interwoven activity and document rules

Because the collaborative writing process is highly document-driven, the activities and documents involved in such a process are strongly interdependent. Indeed, in the process segment of Fig. 3, there are many pairs of complementary activity-document rules connected via dual activities and documents. Notice that activities and documents need not match one-to-one, one activity to one piece of a document.

Let's consider the pair 3 and 4, connected via the activity **write** and the document **paper**. Rule 4, a document-based rule, explicitly gives the structure of the multi-authored document **paper**. The **sections** sub-document itself requires an additional rule as does the **research** task (not included). Rule 3 is an activity based-rule that breaks down the **write** task into several subtasks whose goals roughly match the sections of the written document itself as schematized below:



Of the set of rules, either rule 1 or 2 could be considered the parent rules, since any activity or document can be reached using either as a starting point. However, we can also consider rules 3 and 4 as defining a set of sub-processes that can stand on its own (write the paper, neglecting research, editing, and publication).³ Thus, we could trigger the creation of streams of documents and activities relating to writing the paper by firing either the document rule for **paper** or the activity rule for **write**. For instance, initially triggering rule 4 will create the **paper** document. Since **write** is the activity dual to **paper**, it is then enabled, triggering rule 3. Alternatively, the same chain of events could be initiated from the activity side by triggering rule 3 first.

4.4 Document processes

Taking advantage of the interplay between document and activity rules, we can easily model both decomposable and non-decomposable document processes, whose characteristics were summarized in section 2, and the dynamic mixing of the two. To recap, roughly speaking, decomposable processes permit a concurrent flow of activities associated with document parts, while non-decomposable ones require either a sequential flow of activities or highly coordinated parallel flows of activities.

³ Parallel processes result when several rules are fired independently of each other. For example, there may be many ongoing research projects within a group.

```

|task = paper| --> |task = write| |task = edit/proof| |task = sendToReferee| :: (Rule 1)
|doc = final| |doc = paper| |doc = edits| |deadline = deadline| |
| | | |end = End| |role = copyeditor| |end = End|
| | | |end = End| |end = End|

constraint = {write, edit/proof} precede sendToReferee
constraint = End ≤ deadline

|doc = final| --> |doc = paper| |doc = edits| :: (Rule 2)
|task = paper| |task = write| |task = edit/proof|
| | | |group = copyeditor|

constraint = paper triggers edits
constraint = edits trigger paper

|task = write| --> |task = intro| |task = research| |task = conclude| :: (Rule 3)
|doc = paper| |doc = intro| |doc = concl| |role = authors|
| | | |role = authors|

|doc = paper| --> |doc = intro| |doc = sections| |doc = concl| ::(Rule 4)
|task = write| |task = intro| |length = Lsections| |task = conclude|
|length = length| |group = authors| |group = {authors,}| |group = authors|
| | | |consult}| |length = Lconcl|

constraint = Lintro + Lsections + Lconcl ≤ length

|task = sendToReferee| --> |task = referee| |task = comment| :: (Rule 5)
| | | |role = referee| |doc = comments|
| | | |deadline = deadlineRef| |role = referee|
| | | |end = End|

constraint = End ≤ deadlineRef

|task = referee| --> |task = revise| |task = publish| |task = party!| ::(Rule
|status = success| |doc = revision| |doc = revisedPaper| |role = authors|
| | | |role = authors| |role = publisher|
| | | |deadline = d1| |deadline = d2|
| | | |end = End1| |end = End2|

constraint = revise precedes publish
constraint = publish enables party!
constraint = End1 ≤ d1
constraint = End2 ≤ d2

|task = referee| --> |task = backToTheDrawingBoard| :: (Rule 7)
|status = failure| |role = authors|

|doc = revisedPaper| --> |doc = comments| |doc = revisions| :: (Rule 8)
|task = revise| |task = referee| |task = revise| |
| | | |group = referee| |group = authors|
| | | |parent = paper| |parent = paper|

constraint = comments precede revisions

```

Fig. 3 Collaborative multi-authoring process grammar segment.

The decomposability or non-decomposability of a document process is defined within the context of the activities acting upon the document and its parts. In particular, the decomposability of a document may vary over time, shifting from one end of the spectrum and back again. For example, while the *paper* document is here considered decomposable with respect to the activities of writing the introduction, the main sections, and the conclusion (rule 4), it is not viewed as decomposable with respect to **refereeing** and **revising** (rule 8), nor with respect to **writing** and **editing/proofreading** (rule 2). Fig. 4 highlights the document view of the multi-authoring process

considered here, making salient the dynamic shifts in decomposability. Below we describe in more detail both the decomposable and non-decomposable aspects of this simplified process.

Decomposable processes: parallel activity flow

Rule 4 gives an example of how to handle the decomposable aspects of a document process. Here, we have modeled a *paper* as a document decomposable into three parts: an *introduction*, several *sections*, and a *conclusion*. The absence of structural constraints (such as *precedes* or *triggers*) indicates that these are independent subdocuments and that, in the absence of

timing constraints on their dual tasks, work may proceed on each of these in parallel. In this scenario, version streams of the separate subdocuments evolve independently. Alternatively, we might wish to include structural constraints to represent actual interdependencies. For example, changes in the *conclusion* might need to be taken into account in the *introduction*, and the paper would no longer be entirely decomposable during the writing phase.

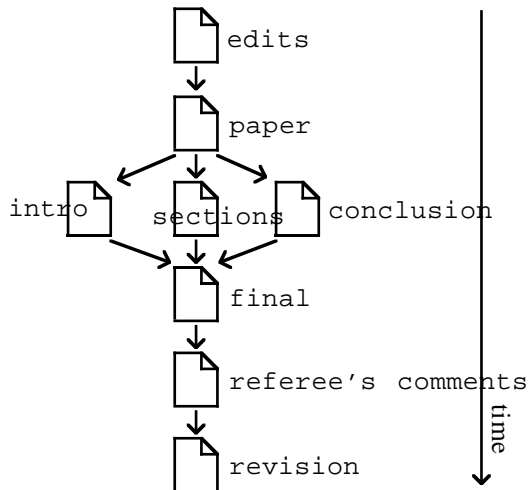


Fig. 4 Document view: dynamics of decomposability.

Non-decomposable processes: sequential activity flow

Rule 8 shows how a non-decomposable document process can be handled by strict sequentialization of activities dependent on the same document through use of a local constraint of the form *doc1* precedes *doc2*. This document rule describes the structure of the final revision to be submitted for publication: the referee's *comments* are incorporated into the parent document *paper* as *revisions* to produce the *revisedPaper*. The local *precedes* constraint enforces that the authors cannot begin revising the paper until they receive the referee's comments, thus enforcing strict sequentiality of activities acting on the same non-decomposable parent document object, *paper*. Both the referee and the authors modify copies of the same document in a strictly coordinated sequence.

Non-decomposable processes: coordinated parallel activity flow

Rule 2 indicates that the *paper* document is also not decomposable with respect to the activities of *writing* and *editing/proofreading*. This rule coordinates concurrent authoring and editing through the two *triggers* constraints: new versions of the *write*-stream of *paper* are made available to the actors responsible for the edit/proof-stream of *edits*, and vice-versa.

5. DISCUSSION

In this paper we have presented a novel process modeling formalism, Generalized Process Structure Grammars (GPSG), and shown how it can be used to create flexible representations of work. The features of GPSG which promote modeling openness are (1) the dynamical generativity of the process grammar approach; (2) the representational freedom accorded by constraints for describing soft dependencies between

activities and documents; and (3) the notion of duality between activities and documents. While in practice, GPSG could be used to encode rigid models of work as readily as flexible ones, in section 4, we have shown that GPSG has the flexibility required to represent a complex multi-authoring process which coordinates the interdependent activities of many people across a common structured document.

5.1 Ongoing work

To support the GPSG formalism, we are developing a simulation prototype called Zippin which guides the modeler in exploring the process space defined by a given grammar. The simulation tool currently shows several views of the process, including: (1) a task view, which shows the hierarchical decomposition of tasks into sub-tasks; (2) a time view, which shows the temporal dependencies between tasks; (3) a role view, which displays a solution for scheduling people to tasks; as well as (4) a text-based grammar editor for entering the process grammar. The views are interactive: changes made by the user in one view propagate to others. In the context of the multi-authoring example of the previous section, for instance, the user could explore the different process alternatives (paper accepted/rejected), move activities backwards and forwards in time, change paper deadlines and page limits, increase/decrease the number of authors or copy-editors, add/remove dependencies between different sections of the paper, and so on. All the while, the simulator ensures that these changes lie within the process space carved out by the rules and the constraints of the multi-authoring process grammar. For example, if the number of authors are too few compared with the estimated duration of activities to finish writing the paper by the deadline, the simulator will warn the user.

The current implementation of Zippin does not yet, however, recognize document-based rules as well as task-based rules. In addition to providing this functionality, we are also working towards improving the different views by increasing their interactivity and by constructing a higher-level grammar view that makes the GPSG formalism accessible to the typical user. In addition, we are developing new views, for example, a document view, which tracks the passage and evolution of documents through the work process and a process view, which provides an overall graphical view of the process grammar. Our long-term goal is to build a full-fledged distributed tool for simulating process enactment, which (1) can be used by a group of people to collaboratively design and test work processes; and (2) can be integrated with a workflow engine which enacts the process instance descriptions generated by the simulation and modeling tool.

Such a simulation and execution environment will be quite different from that of traditional workflow systems. The Zippin generator *qua* simulation tool will allow users to explore different process plans (i.e., different legal phrases of the grammar) and to evaluate them. Once the user has chosen a plan, the constraint solver resolves the constraint set to generate a preliminary schedule and then returns to the user, allowing further

exploration of the space. The user can then instruct the system to construct an executable process specification (analogous to the process instance of a traditional workflow system). If, during execution the user needs to change the process model, then s/he can go back to the simulation environment. The simulation tool will take into account what has already been executed and cannot be undone, and will allow the user to explore the process space from that point onwards. The newly refined plan can then in turn be executed.

From the side of execution, all this assumes a suitable workflow engine which has the capabilities, among others, to support overlapping, long-lived tasks and document versioning and control. Indeed, to take full advantage of the expressiveness of the GPSG approach, a next generation workflow engine is required, and is in fact being prototyped here at RXRC Grenoble on top of the CLF (Coordination Language Facility), a development environment for the coordination of distributed objects [3].

ACKNOWLEDGEMENTS

We are very grateful to J.M. Andreoli, U. Borghoff, S. Castellani, A. Grasso, F. Pacull, P. Rivizzigno, and G. Teege for useful discussions and feedback.

REFERENCES

1. Abbott, K. R. and Sarin, S. K. Experiences with workflow management: Issues for the next generation. In *CSCW'94*, ACM, Chapel Hill, NC, 1994.
2. Ait-Kaci, H., Podelski, A., and Smolka, G. A feature-based constraint system for logic programming with entailment. Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslauten, Saarbrücken, Technical Report March 1992.
3. Andreoli, J.-M., Freeman, S., and Pareschi, R. The Coordination Language Facility: coordination of distributed objects. *Theory and Practice of Object Systems* 2, 3 (1996)
4. Bannon, L. J. The politics of design: representing work. *Communications of the ACM* 38, 9 (1995) 66-68.
5. Bowers, J., Button, G., and Sharrock, W. Workflow from within and without: technology and cooperative work on the print industry shopfloor. In *ECSCW 1995*, Kluwer Academic Publishers, Stockholm, 1995.
6. Bowers, J. and Churcher, J. Local and Global Structuring of Computer Mediated Communication: Developing Linguistic Perspectives on CSCW in COSMOS. In *Proc. of 2nd Conf. on Computer-Supported Cooperative Work*, Portland, Oregon, 1988.
7. Ellis, C., Keddara, K., and Rozenberg, G. Dynamic change within workflow systems. In *COOCS'95*, ACM, Milpitas, CA, 1995.
8. Ellis, C. A. and Nutt, G. J. Modeling and enactment of workflow systems. In *Application and Theory of Petri Nets*, Springer-Verlag, Chicago, Ill, 1994.
9. Florijn, G., Besamusca, T., and Greefhorst, D. Ariadne and HOPLa: flexible coordination of collaborative processes. In *Coordination '96*, Springer-Verlag, Cesena, Italy, 1996.
10. Fuchs, L., Pankoke-Babatz, U., and Prinz, W. Supporting cooperative awareness with local event mechanisms: the GroupDesk system. In *ECSCW'95*, Kluwer Academic Publisher, Stockholm, Sweden, 1995.
11. Gazdar, G., Klein, E., Pullum, G., and Sag, I. *Generalized Phrase Structure Grammar*, Basil Blackwell Publisher Ltd., Oxford, 1985.
12. Gance, N., Pagani, D., and Pareschi, R. *Generalized Process Structure Grammars for Modeling Collaborative Writing*. Rank Xerox Research Centre, Grenoble, Technical Report March 1996.
13. Malone, T. W., Crowston, K., Lee, J., and Pentland, B. Tools for inventing organizations: Toward a handbook of organizational processes. In *Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Society Press, Morgantown, WV, 1993.
14. Pentland, B. T. Grammatical models of organizational processes. *Organization Science* 6, 5 (1995) 541.
15. Pentland, B. T. *Process grammars: a generative approach to process redesign*. MIT-Sloan School, CCS Working Paper 178, 1994.
16. Posner, I. R. and Baecker, R. M. How people write together. In *Proc. of the 25th Hawaii Int. Conf. on System Sciences*, Hawaii, 1992.
17. Robinson, M. Design for unanticipated use... In *Proc. of the 3rd European Conf. on Computer Supp. Coop. Work*, Kluwer Academic Publishers, Milan, Italy, 1993.
18. Sachs, P. Transforming work: collaboration, learning, and design. *Communications of the ACM* 38, 9 (1995) 36-44.
19. Suchman, L. A. Making work visible. *Communications of the ACM* 38, 9 (1995) 56-64.
20. Suchman, L. A. Office Procedures as Practical Action: Models of Work and System Design. *ACM Transactions on Office Information Systems* 1, 4 (1983) 320-328.
21. Swenson, K. D., Maxwell, R. J., Matsumoto, T., Saghari, B., and Irwin, K. A business process environment supporting collaborative planning. *Collaborative Computing* 1, 1 (1994) 15 - 34.
22. Zisman, M. D. *Representation, Specification and Automation of Office Procedures*. PhD Thesis, University of Pennsylvania, Wharton School of Business 1977.