Collaboration Flow Management:
A New Paradigm for Virtual Team Support

Jacques Lonchamp
LORIA, BP 239, 54506 Vandœuvre lès Nancy Cedex, France
jloncham@loria.fr

Abstract. Collaboration flow management is a new paradigm for virtual team support which departs from the classical ‘workflow management’ and ‘collaborative computing’ paradigms. The aim is to support the opportunistic flow of collaboration within a distributed project, considered as a living and self-organizing system. Such a flow includes informal and formal, synchronous and asynchronous, task-oriented and project-oriented collaborative sessions. Some of them are elements of model-driven session-based process fragments. The paper defines the paradigm and describes our Java prototype of collaboration flow management system through a realistic scenario.

1 Introduction

A virtual team (VT) is a team distributed across space, time, and organization boundaries, and linked by webs of interactive technology. As every team, a VT is a group of people who work interdependently with a shared goal. Teams exist for some task-oriented purpose, and differ from social groups (virtual communities, electronic groups) and decision groups (virtual government, direct participation). The growing importance of the VT concept is related to several well known reasons: the diversification of organizational structures towards more distributed, decentralized, and flexible organizations, the globalization of the economy, with the development of worldwide organizations and projects, the rapid growth of telecommunications and Internet in particular. Cross-organizational VTs, such as virtual enterprises, or networked projects, are the most in need of new ways to work and technology-support infrastructures. VTs can improve work performance by reducing costs (cutting travel costs and time), shortening cycle time (moving from serial to parallel processes), increasing innovation (permitting more diverse participation, and stimulating creativity), and leveraging learning (capturing knowledge in the natural course of doing the work, gaining wider access to expertise, and sharing best practices).

Communication tools, especially those aiming at geographical distribution transparency, such as video/audio conferencing systems, are obviously useful for VTs but not sufficient. The success of a VT depends on a set of psychological, organizational, and cognitive factors [11]. A computerized support for VTs should take into account these factors and include them in its basic requirements. First, a sense of identity and
membership is necessary for VTs. Periodically performing informal virtual meetings is not sufficient for developing a collaborative attitude. Participants must feel that the supporting environment helps them to do important and constructive things together, in synergy (R1). Secondly, traditional authority is minimized in VTs, which develop an inner authority based on competencies. In VTs, power comes from information, expertise, and knowledge, not from position. The important things done together through the supporting environment should include expression of competencies, externalization of knowledge and mental models related to the task in hand (R2). Third, trust is the key to VTs. People work together because they trust one another. People make deals, set goals, and lend one another resources. Without face-to-face cues, and the nuances of person-to-person interaction, trust is harder to attain. But trust can be built with the recognition of the contribution that everyone makes, the clarity of positions and commitments (R3). Finally, project and process management is a critical ingredient for successful distributed work. Co-located teams can quickly clarify goals, correct misunderstandings, and work through problems. VTs need to be more explicit in their planning and their processes (R4). When considering these requirements, we claim that no existing paradigm for cooperative systems, neither the ‘workflow management paradigm’ nor the ‘workgroup computing paradigm’, is satisfying. A new paradigm is required for VT support. The second section of this paper elaborates this idea and defines such a paradigm, called the collaboration flow management paradigm. The third section describes our current Java prototype of collaboration flow management system through a realistic scenario. Finally, the last section compare our proposal with related work and draw some conclusions.

2 The Collaboration Flow Management Paradigm

The fourth requirement (R4) stresses the importance of supporting the VT life cycle process. VT projects usually follow a life cycle with an alternation of divergence phases, during which people work individually, and collaborative convergence phases, during which people build some shared understanding, discuss for discovering and solving the divergences accumulated during individual work, and drive the project. Cognitive ergonomists draw a similar distinction for collective design tasks between individual design steps, requiring operative synchronization mechanisms, and collaborative co-design steps, requiring cognitive synchronization mechanisms [4].

A classical Workflow Management System (WfMS) provides support for coordinating individual activities, as those occurring during divergence phases. Generally, a WfMS provides no support for collaborative activities, as those constituting convergence phases. The three first requirements (R1 to R3) imply that a VT environment should mainly support convergence phases. The ‘workgroup computing paradigm’ is not a satisfying solution because it considers collaborative tasks in isolation and does not support the life cycle process as a whole. A computerized environment for VTs should support more or less structured processes whose steps are collaborative sessions either synchronous or asynchronous, informal or formal. On the basis of re-
quirements (R1 to R3), we analyze these sessions as working sessions, during which participants express their views, competencies, positions, and commitments about the task in hand. Cognitive ergonomists distinguish between ‘production activities’, for the collaborative production of artifacts (idea lists, concept graphs, ontologies, design sketches,…), and ‘evaluation activities’, for the collaborative evaluation and integration of individually produced artifacts. We consider also collaborative sessions for project orientation and project organization (objectives, to do lists, plans,…). At a very abstract level, we feel that issue resolution is the basic building block which can be used for describing all these activities. Participants discuss and solve various issues about the artifacts, the shared context, and the collaboration process itself. The computerized environment should support the definition and management of more or less structured processes whose steps are issue-based synchronous or asynchronous, informal or formal, collaborative sessions.

Otherwise, a VT is a living and self-organizing system. For Peter and Trudy Johnson-Lenz [7], post-mechanistic groupware, like all living systems, are ‘rhythmic’ and made of ‘containers’ with flexible and permeable ‘boundaries’. The VT computerized environment should support in particular a range of session (containers) types providing different collaboration rhythms, in accordance with the task urgency and the required depth of issue analysis, and different group boundaries, through the evolving definition of a ‘core team’ and an ‘extended team’. Core team members are strongly involved in the project and use directly the computerized environment. Extended team members should only be involved through standard communication technologies, such as annotating Web pages or participating to Usenet-like forums. Procedures (process models), context, and timing are the three other primitives of post-mechanistic groupware [7]. The choice between these forms (containers, rhythms, boundaries, procedures) should be made dynamically and collectively during the course of the project.

From above, the following four basic functionalities can be identified. (F1) Support different types of short/rapid, more or less formal, synchronous sessions for the core team, such as artifact-centered informal sessions (around a shared document or a free hand drawing) or formal argumentation sessions (with different rhythms and styles of interaction: free, turn talking, circle, moderated). Composite synchronous sessions can mix formal and informal times. (F2) Support longer/slower/deeper, formal, asynchronous sessions for the core team, controlled through the enactment of issue-based and session-based collaborative process model fragments. Such model fragments can specify for instance different structured brainstorming processes, document inspection processes, knowledge elicitation processes, solution evaluation processes (comparative, analytical, analogical). When needed, an asynchronous issue-based session within a process fragment can be transformed into a synchronous session for continuing with a faster rhythm the resolution of conflicting issues. (F3) Support the overall collaboration flow: some asynchronous Project Management (PM) process enables project initiators to launch synchronous sessions and asynchronous process fragments (see Fig.1). This PM process is the spring of the opportunistic flow of collaboration that the VT generates, which can include in particular other PM processes better suited to the circumstances and which replace the initial one (see
section 3.3). (F4) *Play the role of a project memory*, storing all project related information: artifacts, debates, free hand drawings, messages, process trace, etc. These elements should be *externalized* as HTML/XML documents, for easier access and feedback from the extended team during *asynchronous informal sessions* on the Web (see Fig. 2).

![Fig. 1. Main elements of a collaboration flow](image)

<table>
<thead>
<tr>
<th>Session types</th>
<th>Typical examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronous informal</td>
<td>artifact-centered informal session</td>
</tr>
<tr>
<td>synchronous formal</td>
<td>formal argumentation session</td>
</tr>
<tr>
<td>asynchronous formal</td>
<td>instance of a session type within an issue-based and session-based collaborative process model fragment</td>
</tr>
<tr>
<td>asynchronous informal</td>
<td>informal annotation of 'externalized' artifacts using the Web</td>
</tr>
</tbody>
</table>

![Fig. 2. Summary table of the four basic session types with typical examples](image)

### 3 A Prototype of Collaboration Flow Management System

In this section, we describe a scenario supported by our current prototype of *Collaboration Flow Management System* (CfMS). Our lab plans an extension of its building. A VT has been created for discussing different aspects of this project with the point of view of the future residents. The team includes representatives of lab members and participants coming from the university, the town, and the firm of architects. A kick-off informal 'orientation meeting' has first planned a set of virtual meetings about several specific questions. These synchronous sessions are organized by the project manager through the asynchronous PM process (see section 3.3).

#### 3.1. Synchronous Session Support

The session described here is about car parking near the lab. Such a synchronous session does not follow any predefined execution path: the session moderator can describe the problem and organize the work by using the audio channel. In our scenario, the problem is first informally discussed through textual or graphical annota-
tions on the overall building plan (see Fig.3). Social protocols, such as author identification with colors, can be negotiated. Written clarifications may be obtained through the chat tool. In a second time, emerging controversial issues can be formally discussed with pro con arguments, for instance by using a ‘turn talking’ policy (‘free’ and ‘moderated’ policies are also available). On the basis of the resulting argumentation trees (described in more details in section 3.2.2), the session moderator can select some solutions and measure the consensus within the core team with the voting tool. All produced documents (drawings, annotated documents, argumentation trees, vote results) can be saved as HTML documents for feedback from other lab members.

Fig. 3. The synchronous session support, with a formal argumentation (1), a vote result (2), an informal discussion (3) about the freely annotated architectural plan of the lab (4).

3.2. Model-Driven Collaborative Process Fragment Support

Later in the course of the project, the VT has planned an asynchronous model-driven process fragment for a systematic study of the existing cafeteria dysfunctions and for proposing a consensual list of possible causes and solutions for the new cafeteria.

3.2.1. The Meta Model

The process modeling language is defined by a meta model, which extends the classical process view and organizational view of WFMSs with a decision view and a knowledge view (see Fig.4). The process view mainly includes ProcessFragment and Session types. Each process fragment instance is structured into a network of session instances (also called ‘phases’), with precedence relationships, and special phase instances corresponding to the classical workflow operators (AND split, OR split, AND join, OR join phase types). The knowledge view mainly includes Artifact, Component, and ApplicationTool types. Artifact types specialize generic types such as Text, List, Table, Graph (concept map), or Image. Component types specialize ge-
generic types such as ListElement, GraphNode, or GraphVertex. ApplicationTool types mirror the specialization of artifact types, with specializations of generic types such as TextViewer, ListViewer, TableViewer, or GraphViewer. Each Session type grants access to some ApplicationTool types. The organizational view mainly includes Role and Actor types. The decision view describes collaborative work at a fine grain level. Each Session type is defined internally by a set of Issue types, which must be solved either individually or collectively. Each Issue type is characterized by a set of Option types, which describe the possible solutions. At run time, one option is chosen through an argumentation process. Each Option type can trigger, when it is chosen, some Operation type, which can change some Artifact or Component, or change the process state (e.g. termination of a Session), or change the process definition itself.

Fig. 4. The basic concepts of the meta model

3.2.2. The Scenario
A process model fragment is selected and instantiated (see section 3.3). The fragment implements the DIPA approach [10] which formalizes collaborative problem solving for synthesis situations (design of solutions), and for analysis situations (diagnosis). A typical DIPA-based analysis process includes three steps: first, the problem is described and data ('symptoms') are collected; secondly, data are used to find interpretations, i.e. possible causes; lastly, interpretations are used to devise solutions that serve as reparations suppressing the causes of the symptoms. The process model fragment includes three session types. The ProblemDefinition session type contains issues types for defining the problem (individual), proposing a symptom (individual), and terminating the phase (collective). The ProblemAnalysis session type contains issues types for proposing a cause (individual), linking a cause to a symptom (individual), challenging -keeping or discarding- a cause (collective), adding new symptoms
(individual), and terminating the phase (collective). The ProblemResolution session type contains issue types for proposing a solution (individual), linking a solution to a cause (individual), challenging a solution (collective), and terminating the phase (collective). The instantiated process is a sequence of one phase of each type.

When a user, say Jack, logs in a process fragment where he plays a role, he can use the What can I do? guidance query and the What's new? awareness query (for knowing elements that have changed since his last logout). He can also browse various textual and graphical representations of the model types and the process instances. All participants only act by solving typed issue instances. If Jack wants to propose a new cause, he raises an individual ProposeCauseIssue. Then he has to explain his intention as an argument. This individual issue is automatically solved and triggers an operation which creates a new Cause component. If Jack wants to challenge a cause given by Peter, he raises a collective ChallengeCauseIssue. Then, he gives an argument in favor of the Discard option. Other participants can react (support the Keep option or challenge Jack’s argument). The debate takes the form of an argumentation tree, possibly with qualitative importance constraints between arguments. Such a constraint (‘more/less/equally important than’) opens a sub issue, called a ‘constraint issue’, for debating about the value of the constraint. Participants can access various representations of argumentations, like a threaded discussion form (Fig.3) or a graphical form with a ‘playback’ facility (Fig.5). At each argumentation move, the system computes a preferred option by using an argumentative reasoning technique close to the approach proposed in Zeno [5]. The system provides also statistical charts about the participation, the favorite option of each participant, and the ‘level of conflict’, based on the number of changes of the preferred option. Suzan, who plays a role giving the capability to solve this type of issue, terminates the debate by choosing one option. The issue resolution triggers an operation that either keeps or discards the challenged cause. Different tools, like table or graph viewers, are available to the session participants for accessing the process artifacts, such as the symptom/cause/solution concept graph of Fig.5. The process guidance, the argumentation and decision-making support, the awareness facilities, and the way to transform an asynchronous debate into a synchronous one are described in more details in [12], which also gives some information about the current Java implementation.

A model fragment can include ‘open points’ for refining the model at run time. In our model an open point is an instance of a model refinement phase type including a model refinement issue type, with one option type for each possible refinement. At run time, one of these options is chosen, either individually by an actor playing the role of project manager, or collectively by the core team. The resulting operation deploys the refinement by instantiating new phase instances and new phase precedence relationships. By this way, artifact production and process refinement are performed similarly.

3.3 The Project Management Process

In our scenario, the initial PM process follows a very simple model with a PM phase followed by a refinement phase. During the PM phase a participant playing the role of
ProjectManager can plan synchronous sessions and asynchronous process fragments by solving CreateSessionOrProcess individual issue instances. Each resolution adds a new description (name, date, objective, participants, status,...) to a ProjectPlan artifact and calls people for participation through email generation. This planning activity can also result from synchronous ‘orientation meetings’ with the core team. The ProjectManager can also publish in a ProjectRepository artifact, HTML documents produced by the terminated processes/sessions (PublishResult issue type). A dedicated tool for accessing each document is generated and appears in the tool list of the PM session (see Fig.6). The refinement phase enables the project manager (possibly after a debate with the core team) to change the PM policy by instantiating a new PM fragment. For instance, a richer model can include an issue type for publicizing individual tasks by publishing them in the ProjectPlan artifact. Participants can commit themselves to do a given task, and their commitments are published in the ProjectPlan. Model-driven process fragments can also be used within a PM session for defining and installing dynamically (‘on line’) and consensually new model fragments as it is proposed in [6]. It is perhaps not a realistic approach because participants should be aware of all the details of the modeling approach and language. Another way is to use informal sessions for debating the general characteristics of the process model fragment, possibly around high level graphical descriptions. The model programming and installation is then performed ‘off line’ by model developers using the specialized model development environment (editor, compiler, static analyzer, and instantiation tool [12]).

Fig. 5. The asynchronous process-centered session support, with a graphical view of an issue (1), the DISA concept graph built during the session (2), the action panel (3) with a ‘What Can I do?’ guidance query (4), the notification panel (5), and the statistical analysis of the debate.
Fig. 6. The asynchronous project session support with the project plan (1) and the project repository (2). A dedicated tool is generated for each published result (3).

4 Conclusions and Future Work

Our approach is rooted in a wealth of research projects. Among the most important precursors, we can cite Conversation Builder [8] (for genericity, process model fragments, asynchronous collaboration), Cognoter/Argnoter [17] and gIBIS/rIBIS [3] (for argumentation and concept mapping). Our approach reflects also the evolution, in many application domains, from hard-coded tools to model-based generic environments. It is the case for instance for review/inspection systems, moving from specialized tools such as ICICLE, Scrutiny, etc. (see [15]) to generic environments such as CSR5v3 [18] or ASSIST [16]. Two recent systems have strong similarities with our proposal. First, we share with Compendium [2] the idea of modeling collaborative work through a set of typed issues, called ‘question-based templates’ in Compendium. We share with SCOPE [6] the idea of ‘session based process modeling’. SCOPE roughly coincides with our definition of a CfMS. But SCOPE does not provide high level interaction services such as our argumentation service. On the opposite, SCOPE aims at satisfying a larger set of system requirements. For instance, SCOPE provides a specific concurrency control mechanism for enabling sessions to access artifacts simultaneously and for preventing inconsistent changes. In our view, a CfMS only provides the core services for collaboration, such as the management of collaboration-oriented artifacts local to the different sessions. A document sharing system for distributed teams such as BSCW [1] or our Java Motu prototype (available at http://motu.sourceforge.net) should complement the CfMS for managing (sharing, versioning) application artifacts, such as the architectural plans in our scenario. By this way, collaboration is not tied to some particular document management policy.
Similarly, no support is provided for coordinating individual tasks. A cooperative WfMS could complement our CfMS for this aspect.

The ‘collaboration flow management paradigm’ aims at satisfying the basic requirements of VT support, in particular fluidity. As many researchers we have first focused on technical solutions such as structural reflection and meta object protocols (in a early Smalltalk prototype [14]). In the current Java prototype, structural reflection (‘linguistic reflection’ [9]) is not a very practical solution in complex cases which require dynamic coding in the process modeling language followed by a long transformation process (starting with our specific compiler producing Java code followed by the persistency pre processor, the regular Java compiler, and the instantiation process). More realistically, PM process fragments include predefined reflective issue types for simple cases such as modifying the class level relationships of Fig.4 (e.g. adding/suppressing existing issue types or tool types accessibility from a given phase type) or creating new types when the code can be generated from a set of user-provided parameters (e.g. creating new role types). Today, we think that fluidity is a larger problem, with multiform technical and organizational aspects such as fluidity of group boundaries (core team vs extended team), fluidity of roles (in the context of each session), fluidity of containers (sessions) definition and organization (capability to start a formal debate asynchronously and to terminate it synchronously), fluidity in rhythms (between different styles of synchronous interaction), fluidity in formalization (from informal debates to formal argumentation, from to do list to model-driven processes), etc. Our approach deals with a large range of such technical and organizational aspects. Concerning the core services of CfMSs, future work will be directed towards a better support for the VT project memory by using XML technologies. Such a memory should store and restore both in human-readable and machine-readable forms all kinds of information, such as the collaboration artifacts, the process history, the argumentation trees, and even the process model fragments themselves. Another important objective is to provide a larger library of reusable process model fragments, either task-oriented or PM-oriented, and to evaluate them through scenarios and experiments, such as those already performed with our previous prototype [13].

References