A Collaborative Process-Centered Environment Kernel

Jacques Lonchamp

C.R.I.Nancy - Campus scientifique, BP n. 239,
54506 Vandoeuvre les Nancy Cedex, France (jloncham@loria.fr)

Abstract. The Collaborative Process-Centered Environment project (CPCE) aims at applying process modeling approach and technology to a given class of collaborative applications. The challenge is to deal with fine grain entities and interactions, and to provide the high level of adaptability and controlled flexibility required by real world collaborative situations. The concept of a collaborative meta process model which drives the evolution of the executing collaborative process model, and the underlying object-oriented technology are two important aspects discussed in the paper.

1 Introduction

Most software systems only support interaction between a single user and a computer. Even so-called 'multi-user' operating systems and applications basically provide support for isolated work, hiding the activities of other users. In contrast, the general aim of collaborative computing is to suppress the 'protective walls' between users [29], to encourage collaboration, and to directly support and assist the work of groups. Over the past ten years, collaborative computing has established itself as a research field in its own right. Collaborative computing is a complex area because many different shared work styles exist. A fist classification is related to the degree of engagement of participants: for instance, 'division of labour', where several component tasks address separately sub-goals of a common goal, or 'focussed collaboration', where people work closely together. Another taxonomy is the time/location matrix: applications are either local (same place) or distributed (different places) and their interactions occur synchronously (same time) or asynchronously (different times). Other important parameters are the degree of 'repeatability' and 'structuredness' of the collaboration process: from completely unstructured and unpredictable interactions (e.g. a real-time collaborative free-hand sketching tool [11]) to collaborative routines which can be 'programmed' [25].

Early collaborative systems, for instance in the office automation field, have failed because they implicitly assumed a rigid procedural conception of work which is inadequate for representing many real world cooperative work arrangements [30]. They were developed using available computer techniques, especially the dominant procedural programming style. More flexible and customizable approaches have been recently proposed, for instance in the field of software process modeling [14, 15]. Executable software process models are interpreted within so-called 'process-centered environments' (PCEs), to provide control, coordination, assistance, and guidance to the developers. Automation is no longer the central focus, but just one possible

effect of process model interpretation. Flexible programming paradigms (logic, functional, object oriented, rule-based, hybrids) are extensively used [7]. However, most PCEs are devoted to large grain entities management (e.g. design documents, code files), finer granularity being managed by classical tools integrated into the environments. Therefore, PCEs often restrict cooperation to consistent sharing of large grain entities between long transactions. Other styles of collaboration are generally not considered.

The project described in this paper, CPCE (standing for 'Collaborative Process-Centered Environment'), *aims at applying process modeling approach and technology to a given class of collaborative applications.* The challenge is to deal with fine grain entities and interactions, and to provide the high level of adaptability and flexibility required by real world collaborative situations. Not all collaborative applications can take advantage of a process modeling orientation. Asynchronous applications are more likely to be process model driven. They are long term activities, requiring various policies enforcement, and sophisticated assistance: for instance, to 'resynchronize' people working intermittently to the current state of the work through process history and decision rationale. In contrast, synchronous applications are generally short lived, and rely more on spontaneous reactions of the participants sharing a common view of the ongoing work, than on predefined policies and processes enforcement. CPCE project *aims at supporting asynchronous 'focussed' collaborative applications, having a sufficient amount of structure* (see section 2). The environment kernel prototype can be customized for applications belonging to this class (sections 3 to 6 discuss the requirements, design, and implementation).

2 Application Domain

In the class of applications currently supported by CPCE, several participants (local or remote), bring their ideas and opinions in order to build consensually a given artifact. They participate to the work when they wish and freely join and quit. The overall process is long lived, but the elementary activities to obtain consensus or to evolve a specific aspect of the artifact are short lived. The main emphasis is on consensus [23]: most of the decisions about the artifact being designed must be consensually taken all along the artifact construction through issue resolution processes. Issues are solved, either individually by their author or, more often, collectively by the participants through positions (i.e. statements or assertions which resolve the issue), arguments (which either support or object a position), and a resolution protocol implying the selection of a position (e.g. unanimous choice, choice by a majority of participants). In general, all kinds of issue cannot be raised from the beginning: a process including several steps is defined, every step encompassing a subset of the issue type set. Often, the termination of a step is itself an issue to be collectively solved. Parallelism between issues of the same step, and between steps is possible. An issue resolution contributes to (or triggers) a subsequent step which generally evolve the artifact, raising new issues. Every participant plays a given role: a role defines which issues he can raise, which deliberations he can participate in (by giving positions and arguments), which steps he can invoke.

In this paper, the customized environment which exemplifies the approach, supports the collaborative design of a document. Main ideas about the process are taken

from Cognoter [31]. The document design process encompasses several phases.
A brain-storming phase. Participants propose ideas. These ideas are made visible to all participants as soon as they are proposed. As usually during brain-storming, discussion and deletion of ideas by other participants are forbidden during this initial phase "to not interfere with or inhibit the flow of ideas"[31]. Adding a new idea is an issue solved individually by its author, who must provide an argument as the rationale for his proposal.
A structuring phase. Participants propose directed links asserting that an idea should come before another in the document. New ideas can also be proposed during this second phase. The process for proposing links is similar to the process for proposing ideas.
An evaluation phase. Participants evaluate collectively the network of ideas resulting from the two first phases. They eliminate peripheral and irrelevant elements, and fill in missing elements. All issues are solved consensually (e.g. with a majority protocol).
A clustering proposal phase. One participant proposes a set of clusters, and for every cluster, the set of ideas it encompasses. The system computes intra and inter cluster links, on the basis of existing links between ideas.
A clustering evaluation phase. Participants evaluate collectively the proposed clustering. They can evolve it through consensual issue resolutions (e.g. with a majority protocol).
Starting the 'Cognoter-like' design process for a given topic, and finishing every process phase are other collectively solved issues (e.g. with an unanimity protocol).

CPCE team is currently studying a second application in the field of technical review/inspection of software development products [34]. This second application is more complex because it requires various shared work styles: parallel isolated work, for the individual preparation phase, followed by the merging of all individual findings into a common workspace, for the collaborative phase of the inspection. The customized environment will be functionally similar to some recent dedicated environments [12, 18, 22], with a review process not hard coded in the tool but explicitly modeled and tailorable to specific needs and contexts.

3 Main Requirements and Design Decisions

The basic 'process model orientation' of the project means that a set of classical requirements has to be satisfied by the supporting environment, such as: model-based control of user initiatives, model-based automation of some parts of the process, model-based assistance and guidance for users. These aspects have been often discussed for process-centered software engineering environments. For instance, within the ALF project, initiated by the same research team as CPCE [9]. More specific requirements under consideration here are *fine grain interaction modeling, adaptability, and flexibility.*

Fine grain interactions shall be explicitly modeled besides classical process entities such as tasks, artifacts, roles, actors, and their relationships. In the target application domain, it means entities for the description of consensual decisions. Issues, positions, and arguments are frequently used for modeling such deliberations

[16]. More generally, a 'decision-oriented' process modeling is appropriate [13, 17]. A detailed description of the internal structure and semantics of the artifact, which is the topic of most of the deliberations, is also required. CPCE generic model, which is an extension of Potts' model for representing design methods [26], will be described in the next section. Process models and process histories include many objects of various granularities. As persistency of models, histories and rationale is mandatory to ensure model interpretation and retrospective assistance, object oriented repositories are good candidates for founding the supporting environment.

Adaptability has been extensively studied for process-centered software engineering environments. A software process model is built by customizing a generic model, and instantiating it before its execution [6]. The large variety of asynchronous collaborative tasks, sharing an important set of common features, requires a similar approach: the supporting environment shall be a kernel which can interpret every process model customizing a given generic model. The specialization concept, with inheritance for both statical and dynamical aspects, reinforce the interest for an object orientation: generic entity types can capture the common structure and behavior associated to all their instances. For example, what happens when a user gives an argument, whatever its type is. The specific behavior of every customized type is specified at the sub-type level.

Statical customization is not sufficient. Dynamical change to the running process model has been recognized as a major issue by the process modeling community [10]. For collaborative environments two main reasons can be stated: first, groups often evolve and adapt their way of working to their evolutive contexts. Secondly, describing in advance all aspects of a given model is difficult, especially for argumentative entity types such as issues, positions, and arguments. CPCE distinguishes two aspects: (1) the technical aspect of implementing dynamic evolution of the running model, (2) the organizational and decisional aspects of managing evolutive environments. Aspect (2) is one of the main originality of CPCE. The requirement is that the dynamic evolution of a collaborative environment shall be controlled, assisted, and consensual. CPCE solution is to drive process model evolution thanks to another dedicated collaborative process model, called the 'meta-process model'. To avoid meta circularity, changing the meta process model is not required to be itself model driven: the meta process model is statically customized for every application and cannot evolve on the fly. The meta process model is obtained by customizing the same generic model which is used to produce the process model. Both processes are very similar, and participants work in a similar consensual way either to evolve the artifact or to evolve the model which defines how they work. This mirrors usual meetings, where people discuss in the same way of the job and of its organization (see Fig.1). It is worth noting that effective meta process modeling implies fine grain modeling to be able to describe and control the evolution of every fine grained process model component. For aspect (1), the 'full object' orientation of languages such as Smalltalk [4], where all entities, including classes and methods, are dynamically modifiable objects, in conjunction with the interpretative, reflective, and dynamic nature of these languages, make more easy the implementation of the meta level.

Therefore, a persistent object repository extending such a 'full object' language-based environment, and supporting multi-user concurrent access (local or remote), constitutes the core of the CPCE prototype. The object base is used to store the

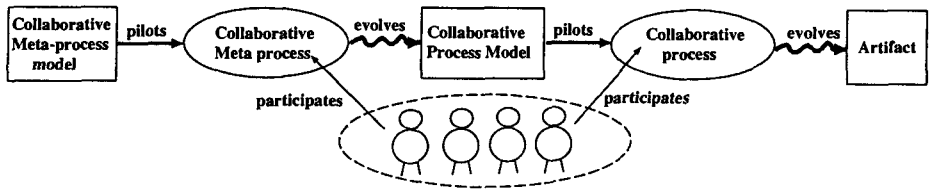

Fig. 1. CPCE logical architecture

artifact, the customized process model, the customized meta process model, the process history and rationale, and the meta process history and rationale. Models are expressed at the schema level, histories and rationales are expressed at the instance level. Users invoke class methods (i.e. methods of metaclasses), either to work (modify the artifact and create new process history instances), or to evolve the process model (modify the process classes and create new meta process history instances).

Dynamic schema evolution, which support dynamic model evolution, has been studied through different perspectives: taxonomy of meaningful changes, semantics of schema changes, and cost. 'Soft changes', which do not require database updates, have been distinguished from more costly 'hard changes' [3, 19]. Here, the meta process specifies which dynamic schema evolutions (i.e. dynamic process model evolutions) are supported and how, on the basis of their significance for the process being modeled, and their practical feasibility in a collaborative setting. Low cost changes are those which can be defined by manipulating menus and typing values, without complex programming. In contrast, changing the code of a method is a costly soft change. The meta process should also enforces integrity rules of the meta model. For instance, adding a new issue class requires at least adding one position class responding to it, and one supporting or objecting argument class.

4 Process Modeling in CPCE

Every customized process model is built by refining the set of predefined generic classes and methods, belonging to the generic model.

From the statical point of view, the generic model depicted in Fig.2 is an extension of Potts' model [26]. New generic classes with regard to Potts' model are written in italics and new link classes are depicted with bold lines. The 'Artifact' class is the root of an application specific hierarchy with application specific semantic links. Every other generic class (e.g. 'Argument') is specialized into generic process model classes ('ProcessArgument') and generic meta process model classes ('MetaArgument'). Then, each of them is further customized according to the needs of a given application. Links between specialized classes and attributes of specialized classes express specific static aspects of the customized model. The set of attributes is richer in the prescriptive model of CPCE than in the descriptive model of Potts. There are both class variables for expressing various model properties (e.g. 'IssueType' specifying the resolution protocol used to solve issues of a given type) and for implementing the relationships of Fig.2 (e.g. argument class X 'ToSupport' position class Y), and

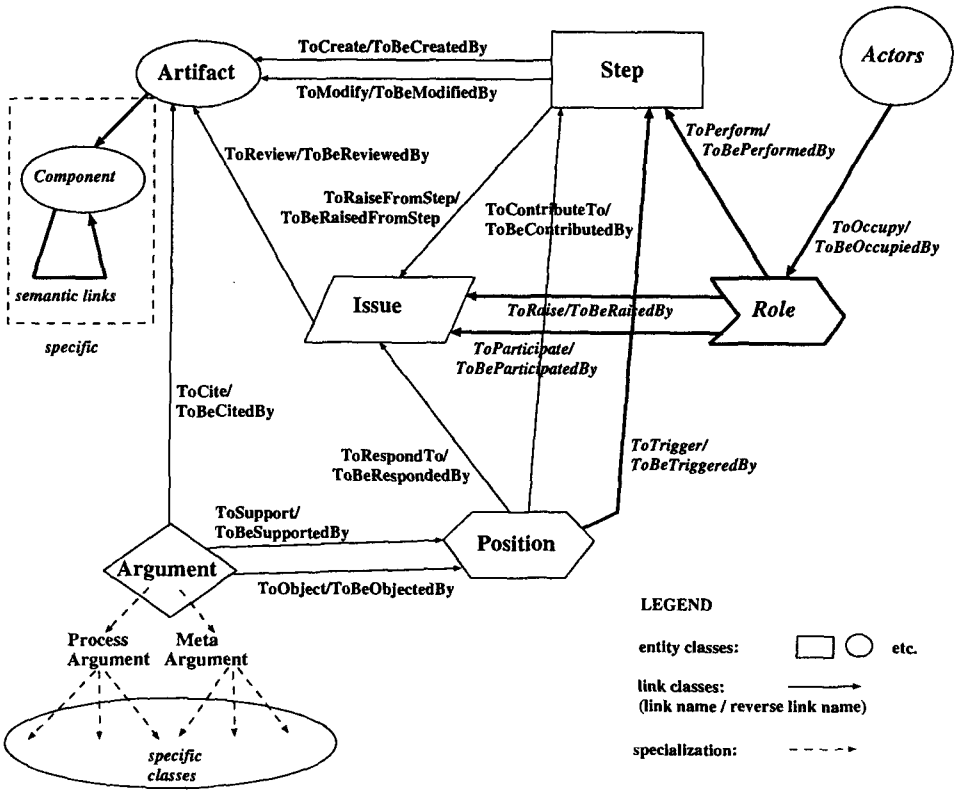

Fig. 2. The generic process model

instance variables for expressing history values (e.g. the text of argument x) and relationships (e.g. argument instance x 'hasSupported' position instance y).

From the dynamical point of view, class methods of generic classes embody the basic behavior of all collaborative environments. They are extended within customized sub-classes. Class variables are extensively used to describe model properties in a declarative way. Many dynamic changes to the customized running model are made just by changing the value of such variables. Class methods of generic classes are written to cope with all the anticipated values of these variables. For instance, the 'SolveIssue' class method of every customized issue class uses a 'SolveIssueFixedPart' class method inherited from the generic 'Issue' class, which can cope with all the anticipated consensus protocols. For every customized issue class, a class variable 'IssueType' gives the kind of protocol which is used to solve it. Therefore, one can change the protocol, under the control of the meta process model, just by changing the value of 'IssueType'. It's a good example of 'low cost' change. Conversely, creating a new unanticipated resolution method is much more costly: a non trivial piece of code has to be included within the kernel part.

5 A Customized Environment Description

This section describes the simple customized environment devoted to the collaborative design of a document and gives a short scenario showing both process and meta process activities.

When a participant connects to the environment, he can interact through a menu driven and graphical interface. The menu driven part allows participants to take initiatives and to obtain various assistance and guidance information either related to the process or to the meta process. Various model-based prescriptions are enforced.
Raise a (meta-)issue. The user chooses a given type of issue among the predefined set of customized issue types (e.g. 'AddIdeaIssue', 'EndBrainstormingPhaseIssue' for the process, 'AddArgumentTypeIssue', 'SuppressTriggerIssue' for the meta process) and relates the new issue to a given ongoing step. He provides the text of the issue to be solved. In fact, dynamically built menus only show permitted choices in accordance to the process state and the user's role.
Give a (meta-)position. The user chooses a given type of position among the predefined set of customized position types (e.g. 'Stop' or 'Continue' for 'EndBrainstormingPhaseIssue') and relates the new position to a given ongoing issue. Only permitted choices are displayed. Currently, the kernel supports a simple model of consensus, with mutually exclusive position types, and one or several argument types supporting every position type. For individually solved issues with a single related position class, the position is automatically given by the system.
Give/Remove a (meta-)argument. The user chooses a given type of argument among the predefined set of customized argument types (e.g. 'InsufficientDuration' and 'InsufficientResults' for 'Continue' position type) and relates the new argument to a given position of an ongoing issue. He can also give an explanation text. Only permitted choices are displayed. In contrast with mathematical theories of consensus described in [23], interactions between participants are possible: a participant may give several consistent arguments for the same position in order to react to other participants' arguments. He can also remove his own arguments, if he changes his mind.
Solve a (meta-)issue. Every customized issue type is characterized by a resolution protocol. If the issue cannot be solved, i.e. no position can be selected by the protocol according to the process state, the request is rejected. The current kernel provides three protocols: individual resolution, collective unanimous resolution, collective resolution by a majority of participants. Others could be supported, such as resolution by the author of the issue after obtaining an authorization.
Perform a (meta-)step. Within every process model there are two kinds of steps: process phases (e.g. 'BrainstormingPhase', 'StructuringPhase') defining which issue types can be raised at which moment, and activity steps (e.g. 'CreateDocument', 'AddIdea', 'DeleteLink') evolving the artifact. Apart from the initial step, every step is either automatically triggered or made ready for invocation by an issue resolution. The user interface is used in the latter case, when a participant takes the initiative to perform a step.
Query about ongoing (meta-)steps and (meta-)issues.
Display historical data: the set of existing (meta-)steps, (meta-)issues, (meta-)positions, (meta-)arguments.

Obtain guidance information: about 'raisable' (meta-)issues and 'performable' (meta-) steps in the current process state and according to the participant's role.

The following snapshots exemplify the interleaving of process and meta process activities. As these activities take place asynchronously on several user workstations we show their effects mainly through graphical representations of process and meta process histories. In Fig.3 the 'Step view model' window gives the overall organization of the document design process with a sequence of phases. It is worth noting that non sequential structures are also possible. Links between phases in the graph are just abstractions of links between some positions inside the phases and subsequent phases (steps) they 'ContributesTo' or 'Trigger' (see Fig2). The 'Step graph' window details the 'BrainstormingPhase' model when participants individually propose their ideas for the document. The 'Issue view model' window gives the position and the argument classes related to the 'AddIdeaIssue' (there is only one position type because this kind of issue is individually solved).

The purpose of the scenario is to exemplify the dynamical and consensual creation of a second argument class ('RelevantIdea') supporting 'AddIdeaPosition'. Extending argumentative capabilities is expected to be a rather frequent kind of dynamic change. The 'Step view history' window displays the current history of the process: two ideas have been proposed. 'Issue view history' windows detail the corresponding individual issue resolutions. 'Idea graph' window displays the resulting document design state.

In Fig.4 we have similar windows showing the meta process model with only one phase ('ChangeProcessModel') and several meta issue types within it for evolving the process model. The 'AddArgumentTypeIssue' model is detailed in the 'Meta issue view model' window. The meta process history shows the dynamical creation of the new meta argument type. The meta issue has been solved consensually by the two model performers each giving a 'UsefulType' meta argument. The 'inspect' windows displays the textual definition of the meta issue. In the TEXT field, the three parameters for creating the argument class appear. No other data is needed. This exemplifies what we have called a 'low cost' unanticipated change.

Fig.5 demonstrates the use of this new consensually agreed argument class for creating, through the menu based interface of one participant, the third idea in the document. The central window shows the interaction trace with an example of a system initiative (trigger). The pop-up menus for process execution and process assistance are pinned up on the low part of the picture. The graphical representation of the new 'AddIdeaIssue' resolution and the new document design state are depicted on the right part of the screen.

6 Some Implementation Issues

CPCE prototype is built on top of the GemStone multi-user object-oriented database management system which offer a distributed client-server architecture[4].

For the control aspect, both controls related to the semantics of generic classes, which are coded in their class methods, and specific controls, through the test of predefined class variables of the customized sub-classes, are supported. For instance, every customized step class has a 'Precond' class variable with a conditional block

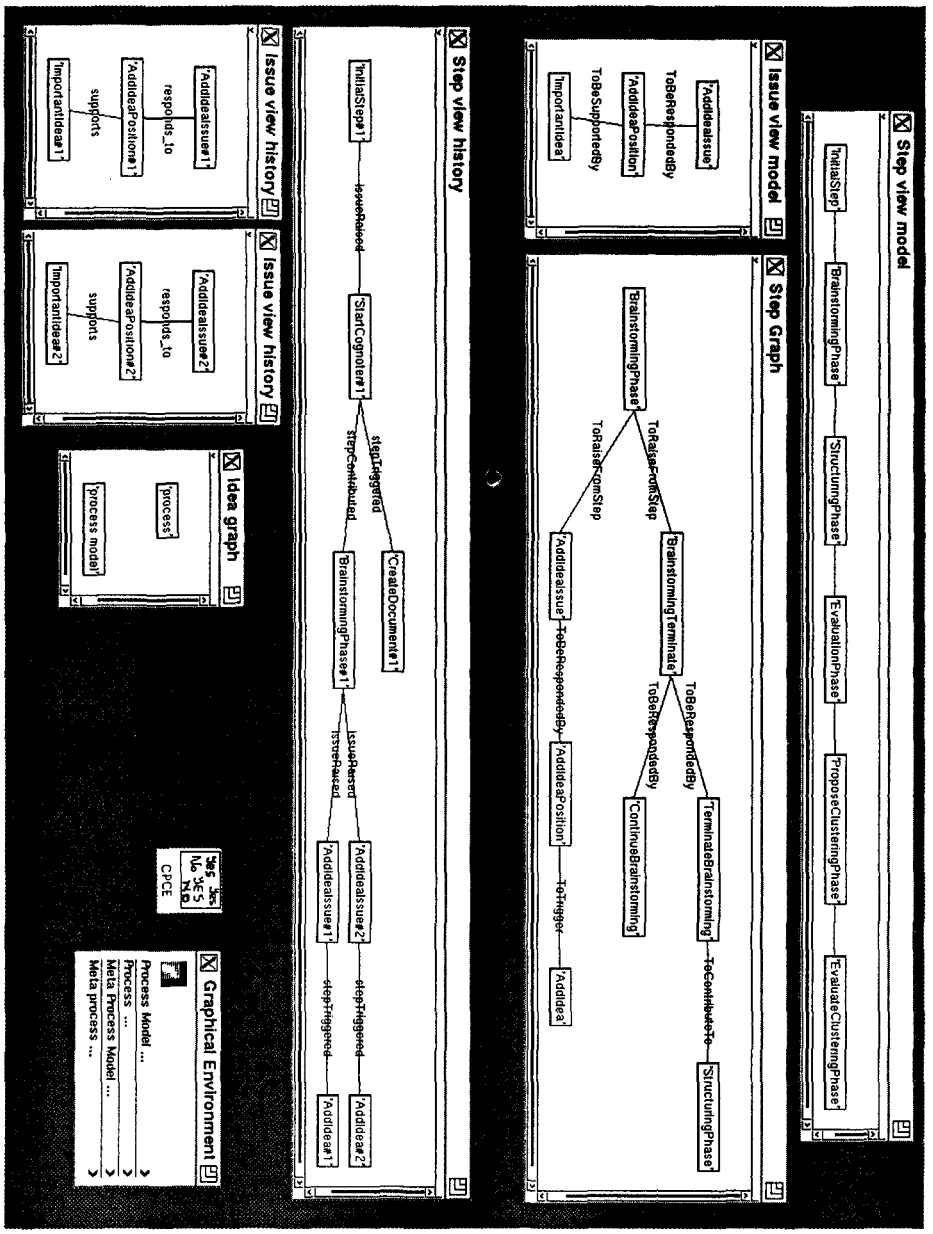

Fig. 3. The process model and process history.

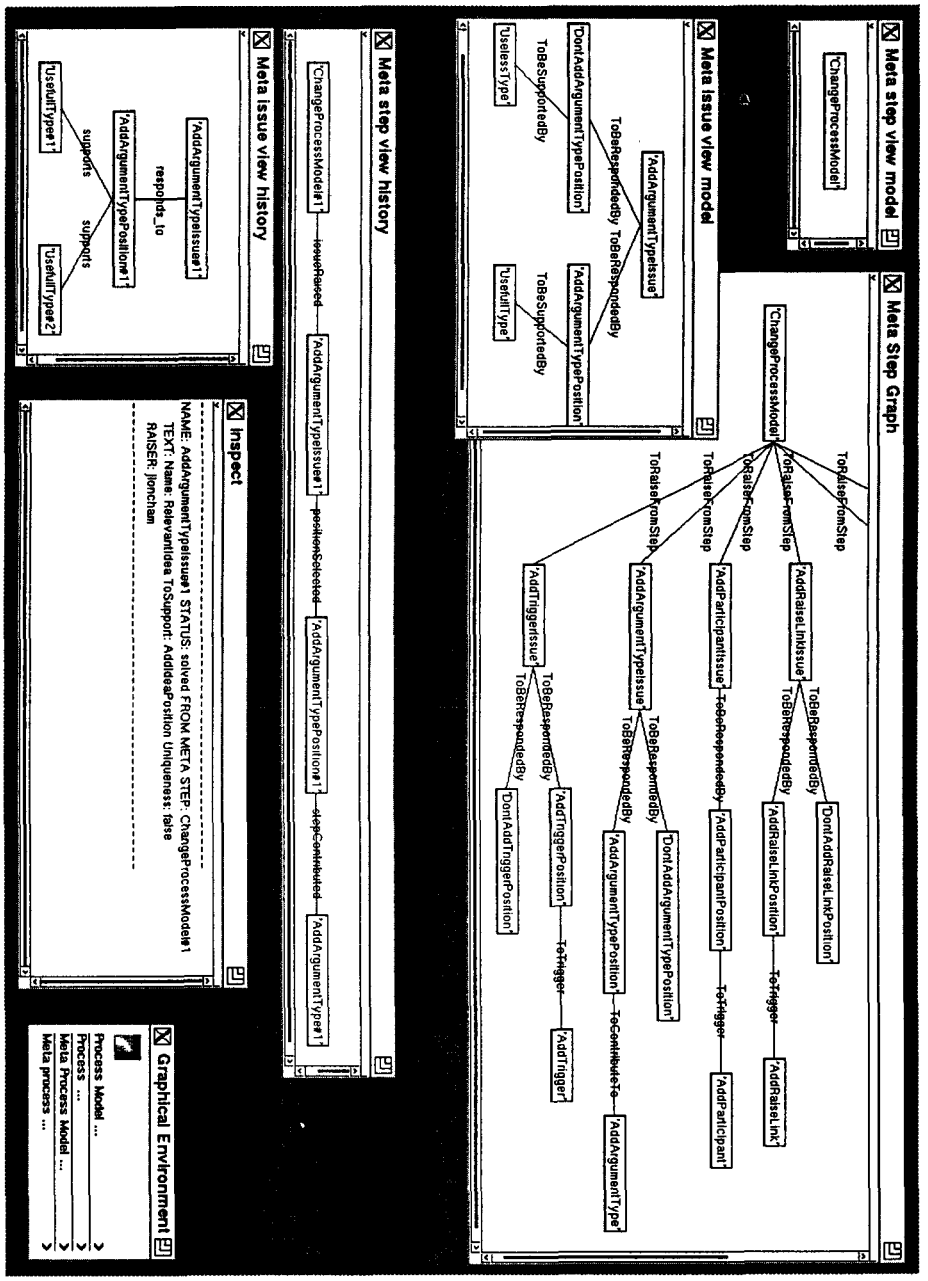

Fig. 4. The meta process model and meta process history.

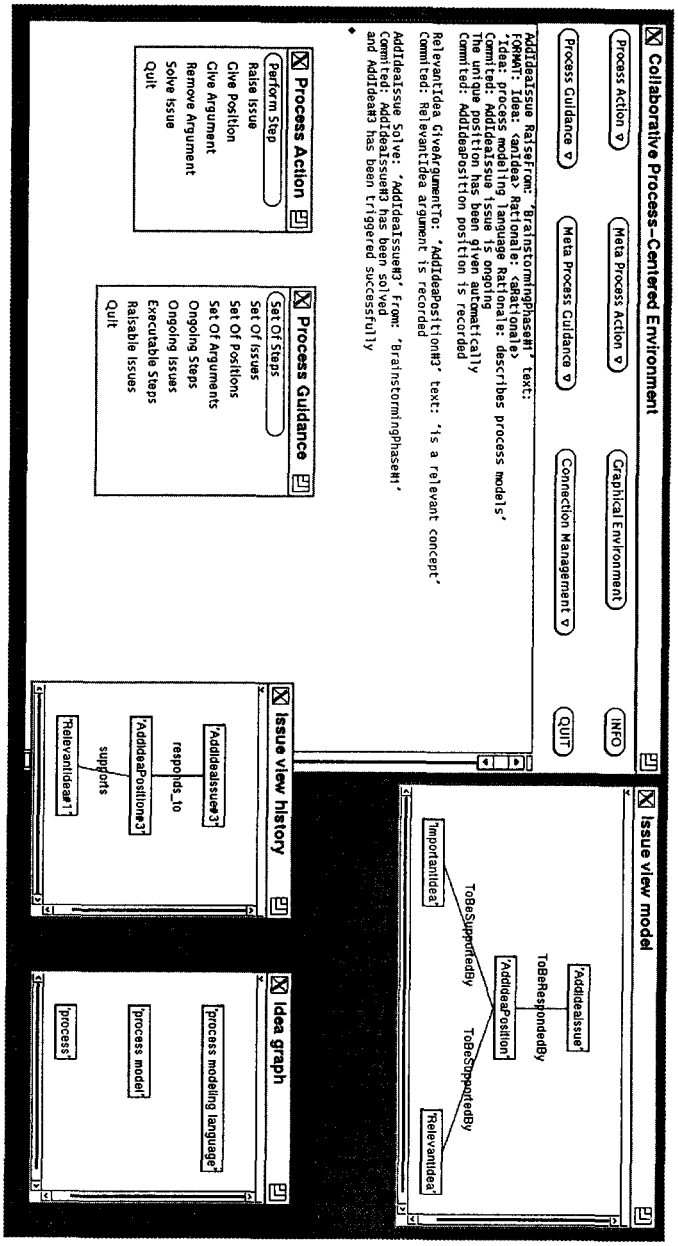

Fig. 5. A participant menu based interface.

(i.e. a parameterized logical expression) as its value. When a 'Perform' message is sent, the method is executed only if the 'Precond' block is evaluated to true.

For the automation aspect, only one kind of process-related 'trigger' is currently implemented. Methods are used to implement triggers [20]. When an issue is solved through a position selection, the generic 'SolveIssueFixedPart' class method tests if a 'ToTrigger' link exists between the customized issue class and a step class. If a link exists, a step of the corresponding class is automatically performed by the system. This implements a simple event(-condition)-action rule. Different events could be considered for other trigger types (e.g. raising automatically a given issue at every beginning of a given step).

For the assistance aspect, the main focus is on 'retrospective assistance' relying on the history and design rationale. Guidance based on the current process state and user's role is also available (see 'raisable issues', and 'performable steps' queries). 'Prospective assistance', for instance through planning and impact analysis capabilities, is not currently considered.

For the evolution aspect, dynamic 'hard changes' rely mainly on class versioning and instances migration as provided by GemStone.

In the document design application, all activities, such as participating to issue resolutions or modifying a given aspect of the document, are short lived. These activities can proceed in parallel and their results are committed into the repository when they finish if no read/write or write/write conflict has occurred between them. In the case of a conflict, a rollback is performed. Obviously this optimistic scheme is not sufficient for all applications. In the next future we plan to enrich the kernel with other schemes. In the technical review application, a new requirement for the kernel is to support parallel isolated work before the merging of all individual contributions into a common workspace for the collaborative phase of the inspection. Sub-schema and sub-database mechanisms are required. Other working modes with semi-isolated work and conflicts resolution should also be supported. It implies to mix asynchronous and synchronous work, for instance for negotiating how conflicts have to be solved. We plan to rely mainly on user consensus to solve conflicts, and to assist them by tracking all dependencies and commitments resulting from their interactions.

The current prototype has shown that simple programming techniques were available for implementing basic control, automation, assistance, evolution, and multi-user support. They will be used more extensively and enhanced in next versions of CPCE.

7 Related Works

Most of process-centered software engineering environment prototypes provide to a certain extent control, assistance, and automation. For instance, the ALF project [9] has put a strong emphasis on assistance and guidance for its users, mainly through planning techniques Object oriented process model formalisms have been studied, among many other paradigms. The IPSE 2.5 project [33] is a well known example of an object oriented process modeling approach. Model customization through class specialization is one of its basic mechanism [28].

Groupware tools for collaborative document design [24, 2] and for collaborative technical review [18, 22] are available. Their processes are hard coded into the tools. Other less specialized groupware tools are partially process model oriented: the Conversation Builder (CB) system [21] is a representative example of such customizable active groupware tools. CB protocols are roughly similar to process model fragments. No concept can be related to the meta process level. A customized environment for technical review built on top of CB has been developed [12].

The general concept of meta process, is discussed in several papers ('the process of development and evolution of a process model' [6]). A few projects have started to study issues for implementing model driven meta process support: reflective high level Petri nets in [1], schema updating controlled by meta-level operations and incremental replanning of task networks when task types dynamically evolve in [8], model construction from a single base role to a set of dedicated roles driven by a meta process model written in a reflective process modeling language in [32]. But modeling and implementation issues for assisted consensual process model evolutions were unexplored so far.

Future work will improve incrementally the kernel. The second version is under development and a customized environment will be devoted to collaborative review/inspection. The main effort will be to make concrete ideas of 'open' (or 'reflective') object-oriented implementations as described in [27]. Implementation aspects that could evolve will be clearly localized ('reified') within specific distinct meta-objects, with their access and change under the control of the meta-process, playing the role of an active meta-interface [27]. [1]

References

1. S. Bandinelli, A. Fuggetta: Computational Reflection in Software Process modeling: the SLANG Approach. Proc. 15th ICSE, Baltimore, 144–154, 1993, IEEE Press.
2. S. Baydere, T. Casey, S. Chuang, M. Handley, N. Ismail, A. Sasse: Multimedia Conferencing as a Tool for Collaborative Writing-A Case Study. Research Note RN/91/77, University College London, 1991.
3. J. Banerjee, W. Kim, H.J. Kim, H.F. Korth: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco, 1987.
4. R. Bretl, D. Maier, A. Otis, J. Perrey, B. Schuchardt, J.Stein, E.H. Williams, M. Williams: The GemStone Data Management System. in W. Kim, F.H. Lochovsky, ed., Object-Oriented Concepts, Databases and Applications, 283–308, Addison-Wesley, 1989.
5. J. Conklin, M.L. Begeman: gIBIS-A Hypertext Tool for Exploratory Policy Discussion. ACM Trans. on Office Inf. Systems, 4, 303–331, 1988.
6. R. Conradi, C. Fernstrom, A. Fuggetta, R. Snowdon: Towards a Reference Framework for Process Concepts. Proc. 2nd European Workshop on Software Process Technology, Trondheim, Norway, LNCS 635, Springer-Verlag, 1992.
7. PROMOTER: Software Process Modeling and Technology. Research Studies Press, 1994.
8. R. Conradi, M.L. Jaccheri: Customization and Evolution of Process Models in EPOS. Inform. System Development Process, IFIP Transact. A-30, Elsevier Pub., 3–20, 1993.

[1] I would like to acknowledge all the members of the Software Engineering Team in CRIN and all the partners of 'PROMOTER' ESPRIT Working Group for fruitful discussions, and Coumba Diouf Ndiaye for participating to the prototype development.

9. G. Canals, N. Boudjlida, J.C. Derniame, C. Godart, J. Lonchamp: ALF, a Framework for Building Process-Centered Software Engineering Environments, in [7].
10. M. Dowson: Software Process Themes and Issues. in [14].
11. S. Greenberg, R. Bohnet: GroupSketch-A multi-user Sketchpad for Geographically Distributed Small Groups. Proc. Graphics Interface'91, Calgary, Alberta, 207–215, 1991.
12. J. Gintell, M. Houde, J. Kruszelnicki, R. McKenney, G. Memmi: Scutiny-A Collaborative Inspection and review System. Proc. CASE'93, Singapore, 1993.
13. U. Hahn, M. Jarke, T. Rose: Teamwork Support in a Knowledge-Based Information Systems Environment. IEEE Trans. on Software Engineering, Vol 17, 5, 467–481, 1991.
14. Proc. of 2nd Int. Conf. on The Software Process, Berlin, Germany, IEEE Press, 1993.
15. Proc. of First Int. Conf. on The Software Process, Redondo Beach, IEEE Press, 1991.
16. A. Jarczyk, P. Loffler, F. Shipman: Design Rationale for Software Engineering-A Survey. Proc. 25th Hawaii Int. Conf. on System Sciences, 577–586, 1992.
17. M. Jarke, K. Pohl: Vision Driven System Engineering. Information Systems Development Process, IFIP Transact. A-30, Elsevier Pub., 3–20, 1993.
18. P. Johnson, D. Tjahjono: Improving Software Quality through Computer Supported Collaborative Review. Proc. Third European Conference on CSCW, Milan, Italy, 1993.
19. W. Kim: Introduction to Object-Oriented Databases. Computer Systems Series, The MIT Press, 1990.
20. W. Kim, Y.J. Lee, J. Seo: A Framework for Supporting Triggers in Object-oriented Database Systems. Int. Journal of Intelligent and Cooperative Information Systems, 1, 1, 127–143, 1992.
21. S. Kaplan, W.J. Tolone, D.P. Bogia, C. Bignoli: Flexible, Active support for Collaborative Work with Conversation Builder. Proc. CSCW'92, Toronto, 1992, ACM Press.
22. V. Mashayekhi, J. Drake, W. Tsai, J. Riedl: Distributed, Collaborative Software Inspection. IEEE Software, 66–75, 9/1993.
23. K. Ng, B. Abramson: Consensus Diagnosis - A Simulation Study. IEEE Trans. on Systems, Man, and Cybernetics, 22, 5, 916–928, 1992.
24. C.M. Neuwirth, D.S. Kaufer, R. Chandhok, J.H. Morris: Issues in the Design of Computer Support for Co-authoring and Commenting. Proc. CSCW'90, Los Angeles, 1990.
25. L. Osterweil: Software Processes are Software Too. Proc. 9th ICSE, Monterey, CA, IEEE Press, 3–12, 1987.
26. C. Potts: A Generic Model for Representing Design Methods. Proc. 11th ICSE, 217–220, 1989.
27. R. Rao: Implementational Reflection in Silica, Proc. ECOOP91, Geneva, 251–267, 1991.
28. C. Roberts: Describing and Acting Process Models with PML. ISPW4, Moretonhampstead, 1988.
29. T. Rodden, J.A. Mariani, G. Blair: Supporting Cooperative Applications. CSCW Int. Journal, 1, 41–67, 1992
30. K. Schmidt, L. Bannon: Taking CSCW Seriously. CSCW Int. Journal, 1, 7–40, 1992.
31. M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning, L. Suchman: Beyond the Chalkboard-Computer Support for Collaboration and Problem Solving in Meetings. CACM, 1, 32–47, 1987.
32. R. Snowdon: An Example of Process Change. Proc. 2nd European Workshop on Software Process Technology, Trondheim, Norway, LNCS 635, Springer-Verlag, 1992.
33. R.A. Snowdon: An Introduction to the IPSE 2.5 Project. ICL Tech. Journal, 6, 3, 467–478, 1989.
34. G. Weinberg, D. Freedman: Reviews, Walkthroughs, and Inspections. IEEE Trans. on Software Engineering, Vol 10, 1, 68–72, 1983.