Working Paper Series
ISSN 1170-487X

Coordinating Collaborative Work in an Integrated Information Systems Engineering Environment

by John C Grundy, John R. Venable, John G. Hosking, and Warwick B. Mugridge

Working Paper 96/5

March 1996

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Coordinating Collaborative Work in an Integrated Information Systems Engineering Environment

John C. Grundy†, John R. Venable†, John G. Hosking†† and Warwick B. Mugridge††

† Department of Computer Science
University of Waikato
Private Bag 3105, Hamilton, New Zealand
{jgrundy,jvenable}@cs.waikato.ac.nz

††Department of Computer Science
University of Auckland
Private Bag, Auckland, New Zealand
{john,rick}@cs.auckland.ac.nz

Abstract. The development of complex Information Systems requires the use of many Information Systems engineering tools. These diverse tools need to be integrated in order to be effectively used by multiple cooperating developers. In addition, the users of these environments require features that facilitate effective cooperation, such as support for collaboratively planning cooperative work, notification of changes to parts of a system under development (but only when necessary or desired), support for keeping aware of other developers' work contexts, and the ability to flexibly engineer or adapt development processes and methods. We describe an integrated Information Systems engineering environment which includes a work coordination tool supporting these requirements.

1. Introduction

Development of complex Information Systems requires the use of many Information Systems engineering tools. These include CASE tools, databases and programming languages, documentation tools, version control systems and project management tools. Effective use of these tools together requires their integration. Ideally tools should be integrated in several ways, including: having a centralised data repository; supporting control and event flow between tools; having a consistent set of tool user interfaces; and allowing software processes to effectively use the tools together [3].

The use of such integrated tools by multiple developers introduces a further requirement for process integration - tool use needs to be coordinated [18]. Developers need to: collaborate to plan their work; know what other developers are doing or have done at various times and to various things; be informed of changes other developers are making to artefacts they are using or interested in; and need to collaboratively modify their work processes (and work plans) during development to suit the needs of the system under construction. Each of these tasks needs to be coordinated at a high level of abstraction in order to facilitate the development of complex systems by multiple developers.

We have developed a tool integration mechanism allowing a wide variety of IS development tools to be integrated [8, 34, 9]. This supports: data integration, by integrating repositories or linking them together and keeping their data consistent; control integration, by propagating events between tools; and user interface integration, by ensuring a consistent user interface is provided for all tools. We have now developed a powerful process integration mechanism which uses a visual work planning language to specify collaborative work plans and to capture modification histories for plan stages. A tool for constructing these plans is integrated with other integrated IS development tools. This supports the capture and presentation of work contexts, which are associated with descriptions of data changes in a tool, and these work contexts are presented with the changes to other developers, helping to coordinate tool use. Modification of work plans during development is supported, facilitating collaborative planning and method engineering.

The following section gives an overview of related research into IS Engineering Environment (ISEE) integration and tools which try to facilitate collaborative work and coordination of work. We then describe our integrated environment from a user's perspective and explain its need for a work coordination layer. This coordination layer is described, and its use for work planning, capture and presentation of work contexts, and specification of interest in changes is illustrated. We then show its usefulness for collaborative planning and method engineering. The architecture and implementation of our tool are briefly described and our research and current and future work summarised.

2. Related Research

Tools supporting different aspects of IS development can be used in isolation, but this results in much redundancy, an inability to effectively share data, and inconsistent user interfaces [23, 3]. Tools supporting different aspects of Information Systems development need to be integrated into a single ISEE which overcomes these problems [23, 26, 24]. Recent research into integrating such environments has focused on control, data, user interface and process integration [3, 23]. PECAN [25], MELD [15], and the Cornell Program Synthesiser [28] utilise a large centralised database to store shared information with restrictive structure editors allowing modification of views of this data. FIELD environments [26, 27] utilise selective broadcasting of events between Unix tools to achieve limited forms of control and user interface integration. Dora environments [24] integrate multiple textual and graphical views of software

development via a restrictive structure editing approach. CASE tools utilise code generation and reverse engineering but only partially keep design and code consistent [35, 32]. Federated approaches use a database, such as PCTE, spread over several locations and which may also utilise hetrogeneous data [3], but these often lack adequate user interface consistency and collaborative work and coordination facilities.

Many collaborative environments and CASE tools only support low-level editing mechanisms [1], including most Groupware systems [5], Mercury [16], Mjølner [21] and C-SPE [10]. As these systems do not facilitate coordination of work, nor the capture and presentation of work context information, effective collaborative work on large systems is not possible. Some groupware systems support limited group awareness capabilities, such as multiple cursors [29], but these usually only inform collaborators about the work artefacts collaborators are immediately interested in; they do not provide the context of others' work and are unable to delay informing collaborators of changes until a later time, when it is needed.

Process-centred environments utilise information about software processes to enforce or guide development. Examples include Marvel [2], CPCE [20], and ConversationBuilder [17]. These environments usually provide low-level text-based descriptions of work rationale, and often do not effectively handle restructuring of development processes while in use [33]. Computer-Aided Method Engineering (CAME) tools, such as Decamerone [14] and Method Base [30], provide support for configuring development processes and tools to a particular application, but often utilise complex textual specifications and don't facilitate work coordination itself.

Workflow-based systems, such as Active Workflow [22] and Domino [19], attempt to coordinate work by describing the flow of documents between collaborators. The workflow approach has proven to be inadequate for most real-world coordination activities. Exceptions to the workflows usually outnumber cases when they are useful, and the workflows often need to be modified while in use [33]. In addition, such systems usually do not model nor facilitate collaboration on the coordination (planning) activity itself [33]. Workflow systems and process-centred environments often have only a tenuous integration with the development tools, and thus the history and context of work artefact changes is difficult or impossible to obtain and to present to collaborating users.

3. An Integrated ISEE

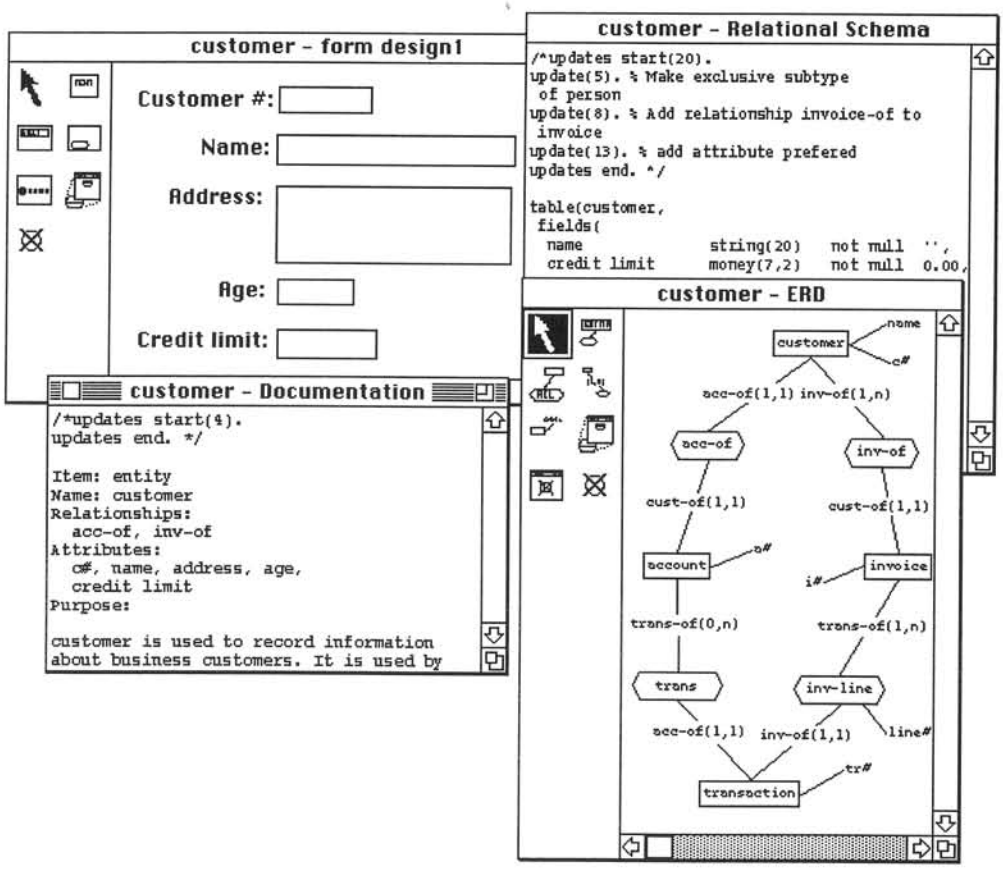

Figure 1. An integrated Information Systems Development environment

We describe an ISEE which incorporates several disparate IS development tools. These include tools for entity-relationship and relational schema modelling [8], form and report design [10], NIAM modelling [34], object-oriented analysis, design and implementation [7], and system documentation [7, 8]. Figure 1 is a screen dump from this environment showing some of the views of IS development it provides.

Window 'customer - ERD' is an entity-relationship diagram showing the major entities and relationships for an invoicing system, and window 'customer - Relational Schema' shows a relational schema for the customer entity. Both of these views are provided by the MViewsER ERD/schema modelling tool [8]. Window 'customer - form design1' shows a form specified for customer based on the customer schema, and these views are provided by the MViewsDP tool for form/report design [10]. Window 'customer - Documentation' shows a documentation view for the customer entity describing its relationships, attributes and purpose. This view is provided by the SPE tool for object-oriented software development, which also provides object-oriented analysis, design and implementation views [7]. Other views supported by our environment but not shown include NIAM views from the MViewsNIAM tool [34], and debugging views provided by the CernoII debugger [7]. We are currently implementing DFD views which will also be integrated into the environment.

It is not sufficient just to integrate tools in terms of their user interface (presentation) [23, 26]. Our integrated tools also support integrated repositories, and information in these repositories and in the tool views is kept consistent when the same or related information in other views is modified. For example, when an entity is modified in an ER view, this change is propagated to other affected views which include this entity information. We utilise the consistency management schemes used within the tools in our environment to also provide inter-tool consistency [7, 8, 10].

A consistency management example is shown in figure 2. The updates made to the customer entity in the ER view are propagated to the schema view and shown as *change descriptions* in the textual view header. These inform a developer of the changes made to the customer entity and that may require updates to the schema definition. Some are easy to make and can be automatically made by the environment (for example, adding, renaming or deleting attributes or changing attribute types). Others are more difficult to automatically implement, such as adding new relationships or changing relationship arity, and the environment leaves these modifications up to the developer. Similarly, changes need to be reflected in the form design view, and these are indicted in a dialog when the view is selected. Some can be automatically actioned, as is shown by the new attribute prefered having an edit box automatically added to the form design, but this still needs designer intervention to move it to an appropriate position (hence the greyed outline).

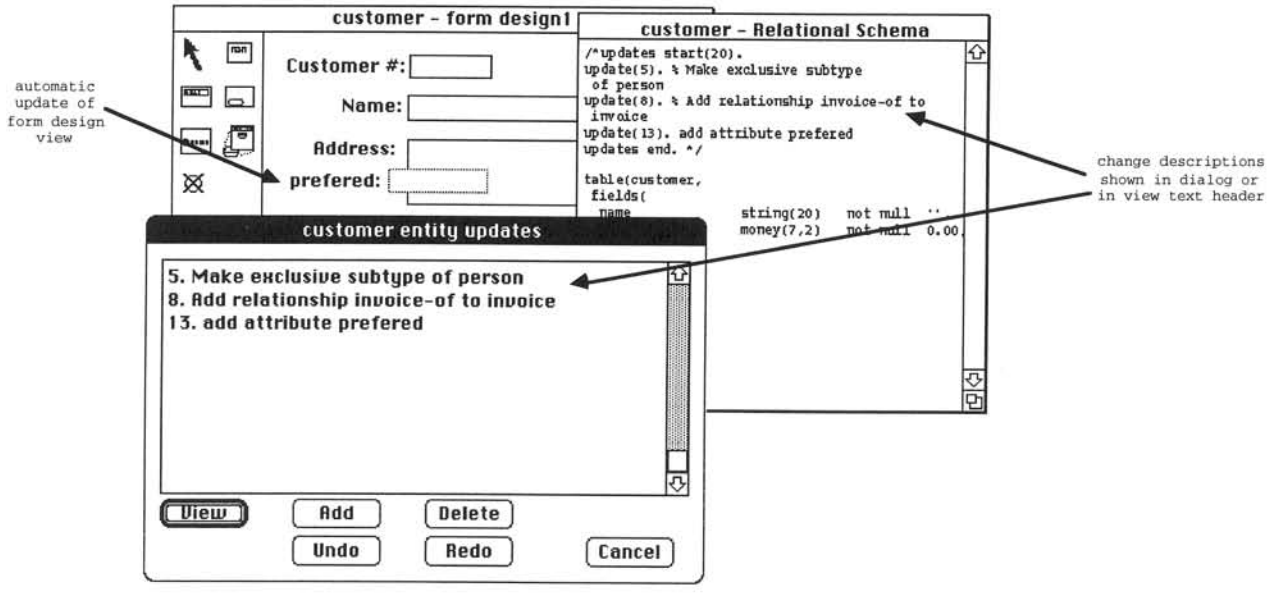

Figure 2. Keeping different views of development consistent.

In addition to the integrated tools and consistency management described above, our environment supports version control and collaborative facilities. Modification histories for all views and software components rendered in views are kept, recording every change made to these artefacts. These histories are partitioned into versions, with version revision, alternates and merging supported. As the storage of these software views and entities is in a shared repository, the versioning facilities support asynchronous collaborative development [11]. In addition, semi-synchronous and synchronous editors are provided for views. Semi-synchronous editing broadcasts change descriptions to all interested

collaborating developers and these are presented to them in dialogues or in view text. Synchronous editing allows developers to simultaneously edit the same version of a view [11].

4. A Work Coordination Layer

While these basic collaborative editing facilities are useful, they are not by themselves sufficient to support large groups of cooperating IS developers. A key problem of systems providing only this low-level coordination of work is the lack of information about the context that work artefact changes have been carried out in [12]. The collaborating user is not told why these changes have been carried out, only the sequence they were carried out in. No support is provided for planning work together nor for grouping changes into histories based on particular tasks and subtasks.

In large CSCW systems, such as collaborative ISEEs, coordination of cooperative work activities is needed [18]. Users must collaborate in the planning of work activities and be aware of the contexts in which other users' work is carried out. Support is needed for defining activities to be done (plans), coordinating the planning activity itself (meta-plans), and restructuring the history of work done to more effectively convey intent ("rewriting history"). The capture of work context and rationale information should be as unobtrusive as possible [4].

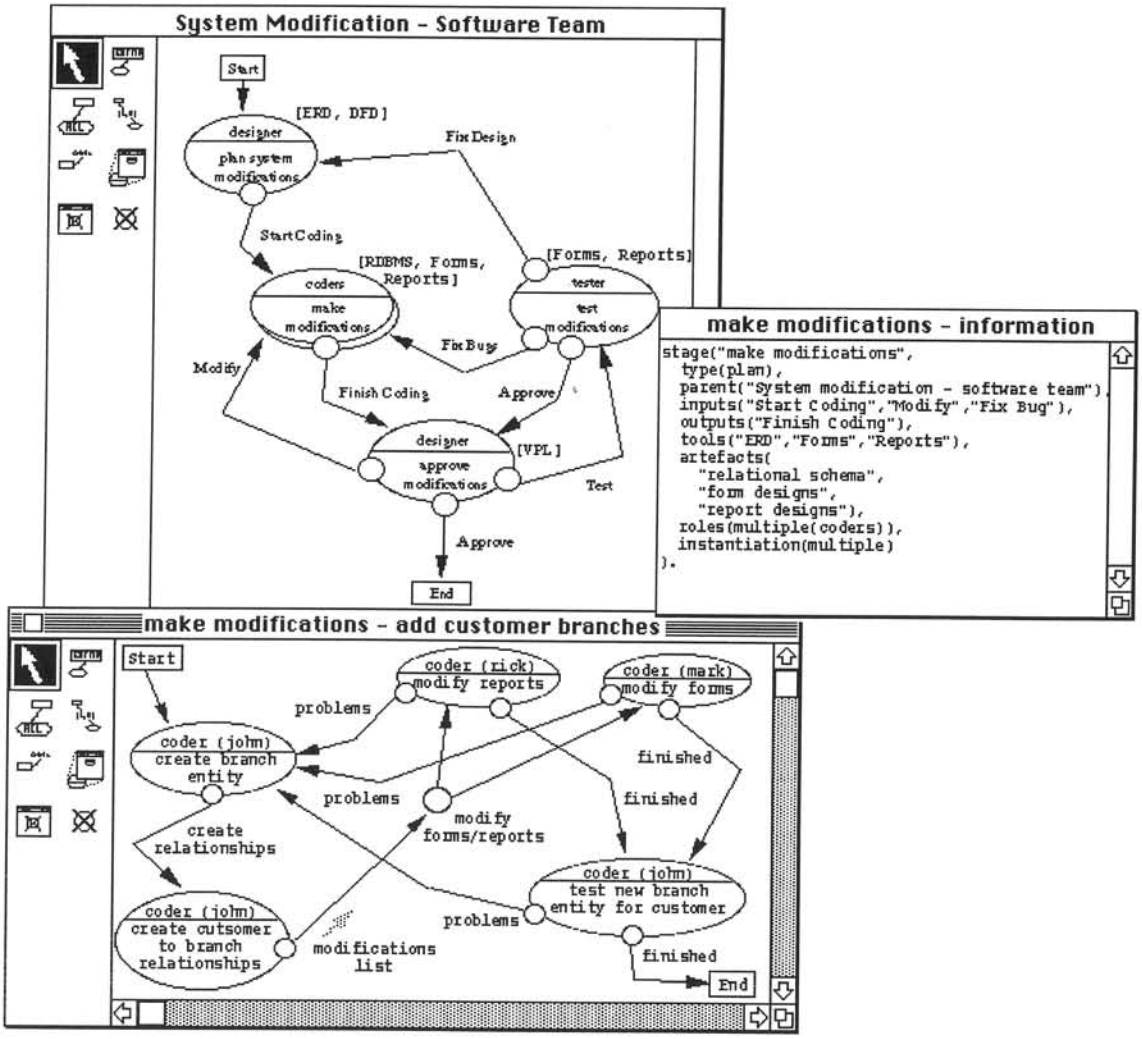

Figure 3. VPL+ views describing some IS development processes.

Rather than the inadequate workflow or process model approaches used by many systems, we have adapted the Visual Planning Language (VPL) [33] to define the context of work and planning activities. VPL allows the definition of plans and subplans for work tasks, and can be used to specify meta-plans for the planning process itself. Useful plans can be abstracted out into policies (basically software process models), which can then be instantiated into further plans. VPL supports flexible restructuring of plans while in use, and partially defined plans can be actioned and later completed or modified. The history of what actually happened when using a plan can be used to restructure the plan to better document

the process and to refine it into a policy. We have extended VPL to produce VPL+, in which a plan stage includes extra information about the work artefacts, CASE tools and collaboration mechanisms used.

Two VPL+ plan diagrams for a software process are shown in figure 3. VPL+ elements include: stages (steps in the process), denoted by ellipses and which include a role and stage description; split stages, which are duplicated for each person involved in the stage; and options, denoted by circles on the stage perimeter, which are used to specify the next plan stage(s). Extra annotations describe tools, work artefacts and communication mechanisms used during and between plan stages. These VPL+ elements are "plan artefacts", which can be modified in the same way work artefacts are modified. Plans are defined by users via synchronous, semi-synchronous or asynchronous editors. Plan modifications can be coordinated via meta-plans (themselves VPL+ plans).

In figure 3, the plan specifies a software process for modifying a software system. A subplan for the "make modifications" stage specifies steps for a particular system enhancement, in this case the extension of the customer entity to include multiple branches per customer. A plan history view is associated with each actioned plan, containing descriptions of changes made to both plan and software artefacts in accordance with this plan. Textual views describing extra information about the purpose of individual plan stages are provided. These views describe which tools are used to carry out the work and which artefacts are used by the plan stage. VPL+ views contain plan artefacts, as opposed to the work artefacts in our ISEE, forming a work articulation layer [31].

5. Capturing and Presenting Work Contexts

The notation for designing plans is used to display and manipulate plans in action. Active stages for a plan are highlighted, options interacted with via menus to advance plan stages and obligations specified between plans and plan stages to ensure users of related plans are informed of changes. The plan view thus specifies the current work context for each member in a collaborating team, and members may have more than one plan view open to see the status of other plans they are interested in or working with. When making work or plan artefact changes, information about the context of this work is captured and presented to interested collaborating users. This information includes who made a change, what was changed, when it was changed, why it was changed and the work or planning context the change was made in. We briefly describe this work context presentation/capture mechanism below. Further details can be found in [12].

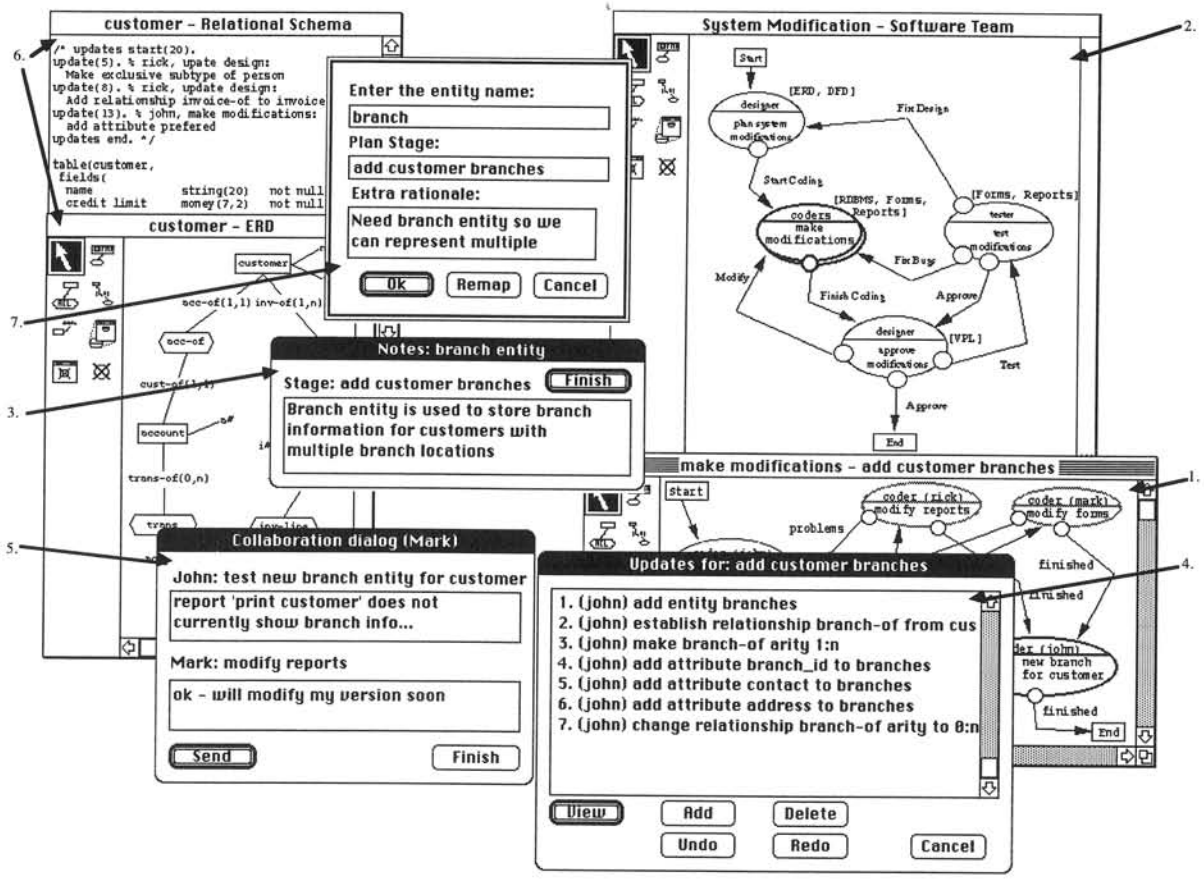

Figure 4. Capturing work context information.

Figure 4 shows how different views can be used to capture work contexts when using the ISEE views from Section 3 (note that a developer would not normally have all of these views open at once!). A "current plan" VPL+ view (1) shows information about a developer's current active stage status (bold border) and indicates the current plans of other collaborators (shaded borders). This captures the current work context for the artefact changes the developer is making. Other shared VPL+ plan views (2) show other plan stages the developer is interested in and also highlight active stages of other collaborators. Highlighting changes when active stages or the shared plan itself change. Artefact notes dialogs (3) specify extra documentation about individual work or plan artefacts, and are shared between all collaborators. Shared modification histories (4) for the current plan stage, or other plan stages the developer wants to review the history of, detail changes that have been made for a task. As plan or work artefact changes are generated, descriptions of these changes are forwarded to the appropriate current plan stage and stored to document the stage's subplan history. These historical changes may also be used to "rewrite history" if the plan is restructured. A collaboration dialog (5) is used for informal, context-dependent dialog between developers. Work artefact views (6) are graphical or textual ISEE editors which capture information about changes made to artefacts. Plan artefact views are manipulated in the same manner, co-ordinated by meta-plans. Augmented artefact dialogs (7) allow a developer to optionally specify extra rationale for low-level work or plan artefact changes. These reasons are stored with the plan stage history and communicated to collaborators.

The current context of work is modified by advancing VPL+ active stages. Incomplete VPL views can be extended as users become more familiar with their actual work processes, unlike many existing workflow and process-centred collaborative systems. Plans can also be restructured while in use, using meta-plans to coordinate this process. Plan history items can be split up if plan stages are split, or moved to other stage histories if plans are deleted.

Collaborating users must be informed of changes to work and plan artefacts they are interested in [12]. In most CSCW environments this amounts to presenting only artefact-level information to collaborators, either directly updating their work artefact views or using version control facilities to indicate changes made by other users. Our approach provides collaborating users not only with change descriptions describing actual work or plan artefact changes, but also with extra information about the work context the changes were carried out in. Presentation of work context information to collaborators is done in a similar way to the capture of work context information. Collaborators share many work and plan artefact views, and will have available the same kinds of views as shown in 4. Some changes a developer makes are directly relevant to their collaborators, such as renaming or deleting entities and attributes, and collaborators should be informed of these immediately. Other changes, such as the addition of new entities, relationships, attributes or forms and reports can be sent for later perusal, as they have more limited effects on collaborators' work. Low-level changes, such as the implementation of procedures, forms or reports not affecting a collaborator's work need not be presented. Collaborators can see from plan histories and various active stages the kinds of activities a developer is doing, and may choose to view these changes or modified artefacts on-demand.

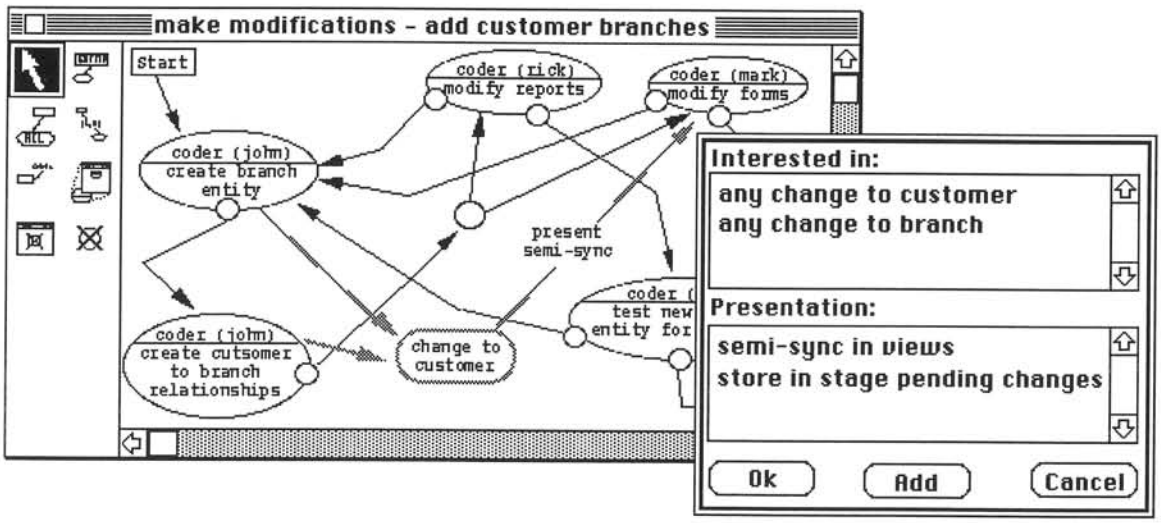

Figure 5. Describing interest in collaborators' work.

Collaborators are informed of changes in various ways. They have similar views to those in Figure 4, including: their own current plan view (1) and other shared VPL+ views (2), both which may indicate other developers' current work contexts, and any change to these contexts (new stage, changes made, change to shared plan(s)). Changes to shared work and plan artefact notes (3) are presented as they are made. Extra comments can be added by a collaborator, which are sent back to other collaborators. Changes to notes are documented in plan histories, and thus can also be viewed asynchronously on-demand. Collaborators can monitor work artefact changes semi-synchronously in a plan history

dialog (4). Informal messages are displayed in a collaboration dialog (5). and these can be responded to as they are received. Shared work artefact views (6) can be edited synchronously, semi-synchronously or asynchronously. Changed items in these views can also be highlighted, allowing collaborators to view the change history for the view or its work contexts' plan histories on-demand. Additional information is presented with change descriptions in work artefact views, including the work context (plan stage) the changes were made in. Hypertext links from these change descriptions allow quick access to plan histories and VPL+ views. Modification histories for individual work or plan artefacts also display the work context each artefact change was made in. Users can thus review change histories by grouping changes by artefact (artefact histories) or by grouping changes by work context (plan histories).

Collaborators need to be able to specify which changes they are interested in seeing and when and how they should be presented [12]. Our extensions to VPL include annotations for specifying obligations between plan stages, to allow users to register interest in particular kinds of plan or work artefact changes, and for defining extra change description processing. Figure 5 illustrates basic annotation indicating which changes to detect. Extra information about how to present these changes to a user can be specified in a dialog. This notation allows users to specify interest in either particular or general kinds of work or plan artefacts. Changes to these artefacts or instances of these artefact types are then communicated to the collaborating users as they occur. Note that many of these interest specifications can be inferred from collaborator plan stage information (for example, if collaborators are using the same artefact and tool but different artefact versions, semi-synchronous editing can be defaulted).

6. Collaborative Planning and Method Engineering

Current Information Systems development methodologies are generally situation-independent. However, researchers have found that due to the increasing complexity of Information Systems, development teams often require methods tailored to particular system development [14]. Our work coordination views assist in Situational Method Engineering [13] by allowing developers to incrementally refine their development methodology and plans. As plan stages record information about the tools to use, artefacts to modify/produce, subsequent plan stages, and also may be exploded into more detailed plans, they facilitate the engineering of software processes in a manner similar to Method Engineering tools.

Our approach has some advantages over comparable notations, such as MEL [14], in that its visual nature is more accessible to developers for visualising and modifying plans than the textual notations of other approaches. As VPL was designed for general work process modelling [33], its high-level nature allows developers to more readily understand and modify process descriptions than text-based process-centred environments or method-engineering tools. It also allows users to modify their plans while the plan is in use, and our VPL+ editors allow users to restructure plans and plan histories after completion so new, improved policies (i.e. software process models) can be developed for reuse.

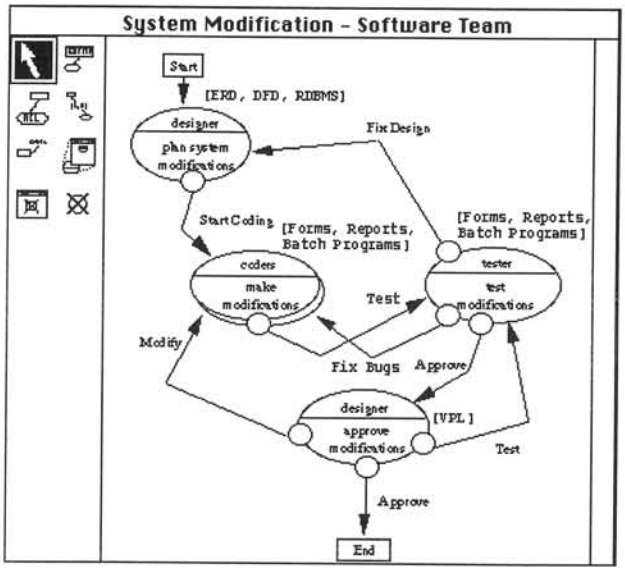

Figure 6. Method engineering techniques.

By providing collaborative editors for VPL+ views, and allowing other VPL+ views to act as meta-plan views (i.e. to plan and record the planning process itself), collaborative planning is supported. Collaborating developers share software process plans and can collaborate on modifying these plans. The use of these shared plans for work context capture and presentation, and specifying interest in changes, allows our VPL+ tools to be used for work coordination, collaborative planning, recording development histories, and method engineering. Figure 6 shows an example of method engineering

with VPL+ views. The System Modification plan (software process model) has now been collaboratively updated to use slightly different plan stages and tools. Database modifications are now done during the "plan modifications" stage (the "RDBMS" tool is now used in this stage and not during the "make modifications" stage), batch programs are now modified and tested, and the subsequent step to "make modifications" is now "test modifications", not "approve modifications" as previously. Any subsequent development using this policy will now use this modified software process.

Our integrated ISEE allows different tools to be used on the same problem domain, with tool data being kept consistent under change and the tools sharing a consistent user interface. Our VPL+ tool allows software processes to be reconfigured during development to better suit a particular development project. Software process models thus evolved can be reused in subsequent development by abstracting them into policies. The specification of artefacts, roles, CASE tools and interest obligations for plan stages gives our integrated environment similar method engineering capabilities to method engineering tools, in addition to its support for work coordination.

7. Design and Implementation

The individual ISEE tools are implemented as a collection of classes, specialised from the MViews framework [6, 10]. MViews supports the construction of Integrated Software Development Environments (ISDEs) by providing a general model for defining software system data structures and tool views, with a flexible mechanism for propagating changes between software components, views and distinct software development tools. MViews describes ISDE data as *components* with *attributes*, linked by a variety of *relationships*. Multiple views are supported by representing each view as a graph linked to the base software system graph structure. Each view is rendered and edited in either a graphical or textual form. Distinct environment tools can be interfaced at the view level (as editors), via external view translators, or multiple base layers may be connected via inter-view relationships.

When a software or view component is updated, a *change description* is generated. This is of the form UpdateKind(UpdatedComponent, ...UpdateKind-specific Values...). For example, an attribute update on Comp1 of attribute Name is represented as: update(`Comp1,Name,OldValue,NewValue`). All basic graph editing operations generate change descriptions and pass them to the propagation system. Change descriptions are propagated to all related components that are dependent upon the updated component's state. Dependents interpret these change descriptions and possibly modify their own state, producing further change descriptions. This change description mechanism supports a diverse range of software development environment facilities, including semantic attribute recalculation, multiple views of a component, flexible, bi-directional textual and graphical view consistency management, a generic undo/redo mechanism, and component "modification history" information.

New software components and editing tools are constructed by reusing abstractions provided by an object-oriented framework. ISDE developers specialise MViews classes to define software components, views and editing tools to produce the new environment. A persistent object store is used to store component and view data.

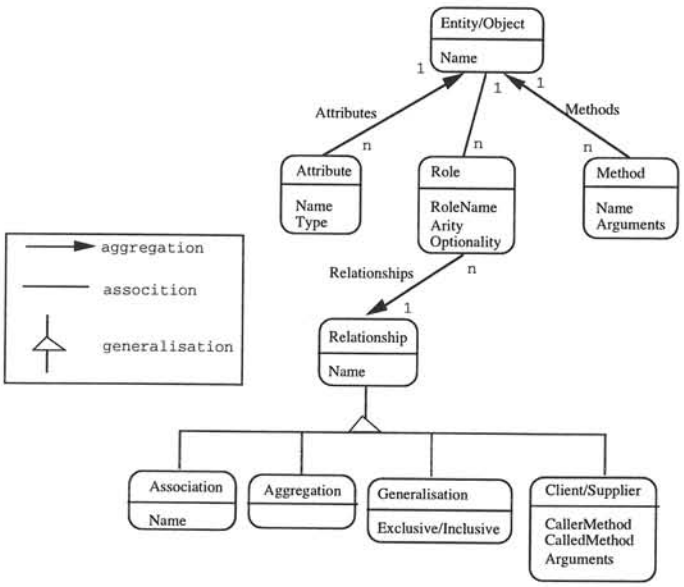

Figure 7. A tool meta-model.

We have developed a technique of integrating tools for multiple design notations using hierarchical, integrated repositories [8]. Figure 7 shows an example of a meta-model of an integrated tool repository for OOA and EERD notations (from [8]). We have recently extended this approach so that information in one tool repository (such as relational schema items) can be linked to similar items in another tool's repository (such as form components).

As these repository items generate change descriptions when modified, inter-repository relationships detect these change descriptions and use them to keep information between the repositories consistent. Figure 8 shows the architecture of our ISEE. Some tools are integrated at the repository level by having an integrated repository [8]. Others have links between repository items which keep related items consistent. Figure 8 shows an example of how data from quite disparate tools is kept consistent: 1) an object attribute is renamed in the OOA/D/P tool, generating a change description. 2) The OO repository is modified and a change description sent to the integrated OOEER tool repository (see [8] for details). 3) the ER tool repository is updated so the OO object's corresponding entity is modified. 4) a change description is sent to the form/report designer repository, updating corresponding form/report fields based on the entity attribute. 5) The form designer views are updated.

We have integrated the SPE OOA/D/P, MViewsER, MViewsNIAM, and MViewsDP tools to produce our integrated ISEE. We are currently implementing a DFD modelling tool and extending SPE to support functional and dynamic models. These behavioural modelling notations will then be integrated and different notations kept consistent using an integrated repository.

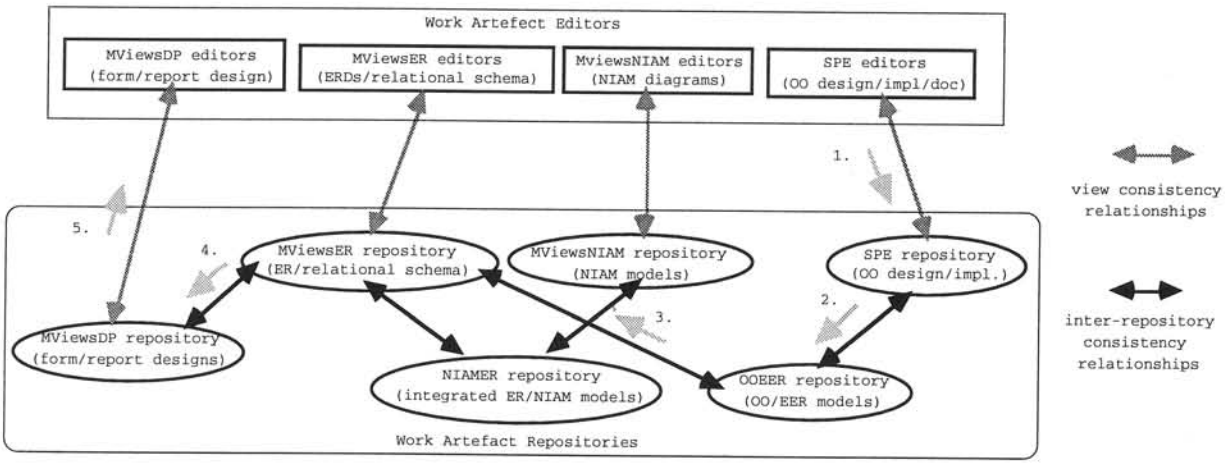

Figure 8. The architecture of the integrated tools.

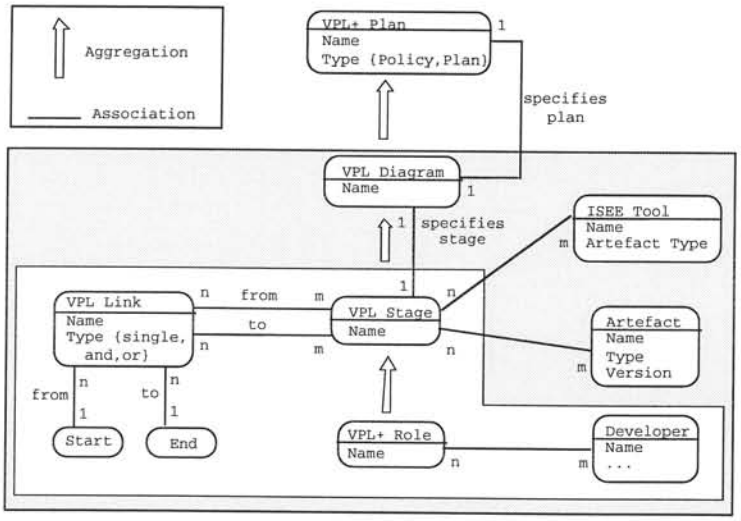

Figure 9. A VPL+ meta-model.

VPL+ views require a repository which defines VPL+ components (plan stages) and detailed information about plan stages (tools used, artefacts, etc.). Figure 9 shows a meta-model of VPL+ describing this information. A VPL+ plan is made up of VPL diagrams, themselves composed of plan stages and links. A plan stage has roles, ISEE tools and work

artefacts. Links can be single (1 stage to 1 stage), or multiple and- or or-dependency links between plan stages. We are currently implementing a VPL+ tool using MViews, which will form the coordination and method engineering layer for our integrated ISEE.

The approach we are taking to integrating our VPL+ tool and ISEE is to have coordination (VPL+) artefacts receive change descriptions from updated work artefacts. Work context information is associated with these change descriptions, then they are propagated to interested collaborators' views for presentation (or they may be stored for later presentation) [12]. Figure 10 shows how ISEE and VPL+ artefacts will interrelate: 1) if a developer modifies a work artefact, a change description is sent to the current VPL+ plan stage. 2) The plan stage augments the change description with work context information, and then forwards this to the work context (current plan stage) of collaborators interested in the change. Additional processing of the change description may also occur here, for example automatically advancing stages, or causing updating of plan or work artefacts. 3) The augmented change description is presented to collaborators in an appropriate manner to inform them of both changes to work artefacts and collaborating users' work contexts, thus facilitating the coordination of work.

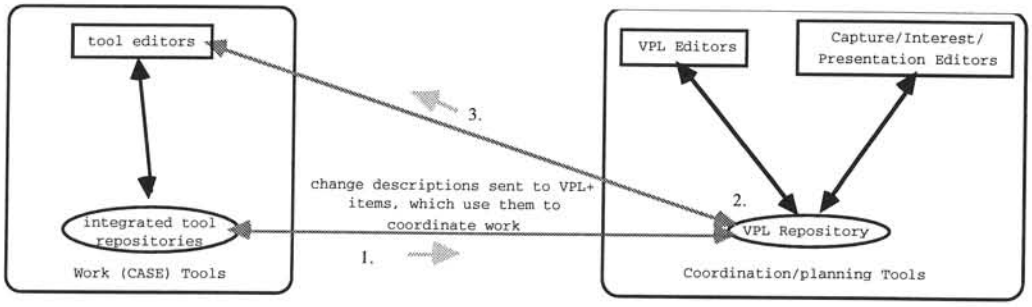

Figure 10. Integrating VPL+ views with integrated tool views.

8. Summary

We have described an integrated environment for Information Systems development which includes work coordination facilities to help manage large systems development. Integrated development tools allow developers to design and build complex Information Systems using a variety of integrated tools, with full data, control and presentation integration between the tools. A coordination layer using a modified version of the Visual Planning Language (VPL) is used to achieve process integration between tools. This includes the ability to coordinate work via work context capture and presentation, collaboratively develop, refine and reuse complex work plans (i.e. software process specifications), and use plans to document development via plan histories.

We are currently implementing VPL+ views to form a coordination layer for our prototype ISEE which incorporates integrated EER, OOA/D, NIAM, relational schema, form and report design, and documentation views. VPL+ views provide a work coordination layer and facilitate collaborative planning and method engineering. We are continuing to extend VPL+ to make it easier for collaborators to specify interest in changes, presentation mechanisms, and to visualise inter-plan stage communication mechanisms, artefact and tool usage, and developer roles. We are also developing a true MetaCASE environment which uses meta-models of notations as CASE tool repository specifications, and allows MViews environments to be more declaratively specified and generated.

References

1. Aean, I., Siltanen, A., Sørensen, C., and Tahvanainen, V.P. A Tale of Two Countries: CASE experiences and expectations. In *Proceedings of the IFIP WG8.2. Working Conference on The Impact of Computer Supported Technologies on Information Systems Development,* Kendall, K.E., DeGross, J.I., and Lyytinen, K. Eds, Minneapolis, June 14-17 1992, North-Holland.

2. Barghouti, N.S. Supporting Cooperation in the Marvel Process-Centred SDE. In *Proceedings of the 1992 ACM Symposium on Software Development Environments,* ACM Press, 1992, pp. 21-31.

3. Bounab, M. and Godart, C. A Federated Approach to Tool Integration. In *Proceedings of CAiSE'95,* Finland, June 13-16 1995, LNCS 932, Springer-Verlag, pp. 269-282.

4. Cockburn, A. and Jones, S. Four Principles for Groupware Design. In *Proceedings of OZCHI'94,* 1994, pp. 21-26.

5. Ellis, C.A., Gibbs, S.J., and Rein, G.L. Groupware: Some Issues and Experiences. *Communications of the ACM 34*, 1 (January 1991), 38-58.

6. Grundy, J.C. and Hosking, J.G. A framework for building visusal programming environments. In *Proceedings of the 1993 IEEE Symposium on Visual Languages,* Bergen, Norway, IEEE CS Press, 1993, pp. 220-224.

7. Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B. Connecting the pieces, Chapter 11 in*Visual Object-Oriented Programming,* Burnett, M., Goldberg, A., Lewis, T., Manning/Prentice-Hall (1995).

8. Grundy, J.C. and Venable, J.R. Providing Integrated Support for Multiple Development Notations. In *Proceedings of CAiSE'95,* Finland, June 1995, LNCS 932, Springer-Verlag, pp. 255-268.

9. Grundy, J.C., and Venable, J.R. Developing CASE tools that support integrated design notations. In *Proceedings of the 6th European Workshop on Next Generation of CASE Tools,* Finland, June 1995, pp. 109-116.

10. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. Support for Collaborative, Integrated Software Development. In *Proceeding of the 7th Conference on Software Engineering Environments,* Nordwijkerhout, Netherlands, April 5-7 1995, IEEE CS Press, pp. 84-94.

11. Grundy, J.C., Hosking, J.G., and Mugridge, W.B., "Supporting flexible consistency management via discrete change description propagation," Working Paper, no. 95/2, Department of Computer Science, University of Waikato, 1995.

12. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D. Coordinating, capturing and presenting work contexts in CSCW systems. to appear in *Proceedings of OZCHI'95,* Wollongong, Australia, Nov 28-30 1995.

13. Harmsen, F., Brinkkemper, S., and Oei, H. Situational Method Engineering for Information System Projects. In *Proceedings of the IFIP WG8.1 Working Conference CRIS'94,* Olle, T.W. and Verrijn, A.A. Eds, Maastricht, 1994, North-Holland, Amsterdam, pp. 169-194.

14. Harmsen, F., and Brinkkemper, S. Design and Implementation of a Method Base Management System for a Situational CASE Environment. to appear in *Proceedings of the 2nd Asia-Pacific Software Engineering Conference (APSEC'95),* IEEE CS Press, Brisbane, December 1995.

15. Kaiser, G.E. and Garlan, D. Melding Software Systems from Reusable Blocks. *IEEE Software 4*, 4 (July 1987), 17-24.

16. Kaiser, G.E., Kaplan, S.M., and Micallef, J., Multiuser, Distributed Language-Based Environments. *IEEE Software* (November 1987), 58-67.

17. Kaplan, S.M., Tolone, W.J., Carroll, A.M., Bogia, D.P., and Bignoli, C. Supporting Collaborative Software Development with ConversationBuilder. In *Proceedings of the 1992 ACM Symposium on Software Development Environments,* ACM Press, 1992, pp. 11-20.

18. Krant, R.E. and Streeter, L.A. Coordination in Software Development. *CACM 38*, 3 (March 1995), 69-81.

19. Kreifelts, T., Hinrichs, E., and Klein, H.K. Experiences with the Domino Office Procedure System. In *Proceedings of the Second European Conference on Computer Supported Cooperative Work (ECSCW'91),* 1991, pp. 117-130.

20. Lonchamp, J. CPCE: A Kernel for Building Flexible Collaborative Process-Centred Environments. In *Proceedings of the 7th Conference on Software Engineering Environments,* Nordwijkerhout, Netherlands, April 5-7 1995, IEEE CS Press, pp. 95-105.

21. Magnusson, B., Asklund, U., and Minör, S. Fine-grained Revision Control for Collaborative Software Development. In *Proceedings of the1993 ACM SIGSOFT Conference on Foundations of Software Engineering,* Los Angeles CA, December 1993, pp. 7-10.

22. Medina-Mora, R., Winograd, T., Flores, R., and F., F. The Action Workflow Approach to Workflow Management Technology. In *Proceedings of CSCW'92,* ACM Press, 1992, pp. 281-288.

23. Meyers, S. Difficulties in Integrating Multiview Editing Environments. *IEEE Software 8*, 1 (January 1991), 49-57.

24. Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R. Dora - a structure oriented environment generator. *IEE Software Engineering Journal 7*, 3 (1992), 184-190.

25. Reiss, S.P. PECAN: Program Development Systems that Support Multiple Views. *IEEE Transactions on Software Engineering 11*, 3 (1985), 276-285.

26. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software 7*, 7 (July 1990), 57-66.

27. Reiss, S.P. Interacting with the Field environment. *Software practice and Experience 20*, S1 (June 1990), S1/89-S1/115.

28. Reps, T. and Teitelbaum, T. Language Processing in Program Editors. *COMPUTER 20*, 11 (November 1987), 29-40.

29. Roseman, M. and Greenberg, S. Groupkit: A groupware toolkit for building real-time conferencing applications. In *Proceedings of CSCW'92,* ACM Press, 1992, pp. 43-50.

30. Saeki, M., Iguchi, K., Wen-yin, K. A Meta-model for representing software specification and design methods. In *Proceedings of the IFIP WG8.1 Conference on Information Systems Development,* Prakash, N., Rolland, C., and Pernici, B. Eds, Como, 1993.

31. Schmidt, K. and Bannon, L. Taking CSCW seriously: Supporting Articulation Work. *Computer Supported Cooperative Work (CSCW): An International Journal 1*, 1-2 (1992), 7-40.

32. *TurboCASE Reference Manual*, StructSoftInc, 5416 156th Ave. S.E. Bellevue, WA, 1992.

33. Swenson, K.D. A Visual Language to Describe Collaborative Work. In *Proceedings of the 1993 IEEE Symposium on Visual Languages,* Bergen, Norway, 1993, IEEE CS Press, pp. 298-303.

34. Venable, J.R. and Grundy, J.C. Integrating and Supporting Entity Relationship and Object Role Models. to appear in *Proceedings of the 14th Object-Oriented and Entity Relationship Modelling Conferece,* Gold Coast, Australia, Dec 13-15 1995, LNCS, Springer-Verlag.

35. Wasserman, A.I. and Pircher, P.A. A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices 22*, 1 (January 1987), 131-142.