In Proceedings of the 7th Workshop on the Next Generation of CASE Tools (NGCT'96), Crete, May 20-24, 1996.

Supporting Collaborative Work in Integrated Information Systems Engineering Environments

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 many Information Systems engineering tools. These diverse tools need to be integrated in order to be effectively used by multiple developers. In addition, these developers require features that facilitate effective cooperation, such as support for collaboratively planning 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 our approach to adding such collaborative work support to an integrated Information Systems engineering environment.

1. Introduction

Development of complex Information Systems requires the use of many Information Systems engineering tools, including CASE tools, databases and programming languages, and documentation and project management tools. Effective use of these tools together requires their integration [3]. The use of integrated tools by multiple developers introduces a further requirement - tool use needs to be coordinated [14]. Developers need to collaborate to plan their work, know what other developers are doing or have done, be informed of changes other developers are making to artefacts they use, and need to collaboratively modify their work processes during development.

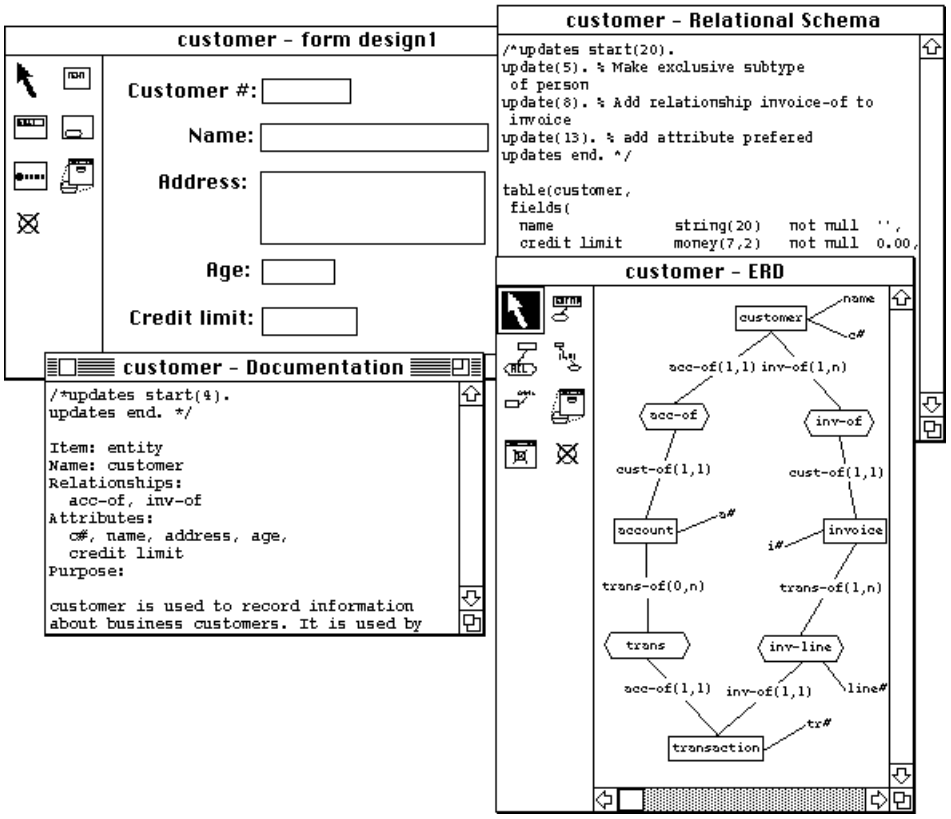
We have developed a tool integration mechanism [7, 22, 8] which supports data integration (by integrating repositories or linking them together to keep their data consistent), control integration (by propagating events between tools), and user interface integration (by ensuring a consistent user interface across all tools). We have developed a process integration mechanism [9] which uses an extended visual planning language to specify collaborative work plans and capture modification histories. A tool for constructing these plans is integrated with integrated IS development tools, supporting the capture and presentation of work contexts. These work contexts are presented with work artefact changes to other developers, coordinating tool use.

2. Related Research

Integrated environments include PECAN [18] and Dora [17], which utilise a centralised database to store shared information. FIELD [19, 20] utilises selective broadcasting of events between Unix tools to achieve limited integration. CASE tools utilise code generation and reverse engineering but only partially keep design and code consistent [23]. Federated approaches use hetrogeneous databases [3], but often lack collaborative work facilities.

Many collaborative environments and CASE tools support low-level editing mechanisms [1], including most Groupware [4], Mjølner [15] and C-SPE [10], but do not usully facilitate coordination of work. Process-centred software engineering environments (PCSEEs), such as Marvel [2] and ConversationBuilder [13], utilise information about software processes to enforce or guide development. PCSEEs usually provide low-level text-based descriptions of work rationale, and often do not effectively handle restructuring of development processes while in use [21]. Computer-Aided Method Engineering (CAME) tools, such as Decamerone [12], provide support for configuring development processes and tools to particular applications, but do not facilitate work coordination. Workflow-based systems, such as Action Workflow [16], attempt to coordinate work by describing document flow. This has proven to be inadequate for most coordination activities, as exceptions to workflows usually outnumber useful cases, and workflows often can't be modified while in use [21]. Most PCSEEs and Workflow systems are not well integrated with existing development tools.

3. An Integrated ISEE



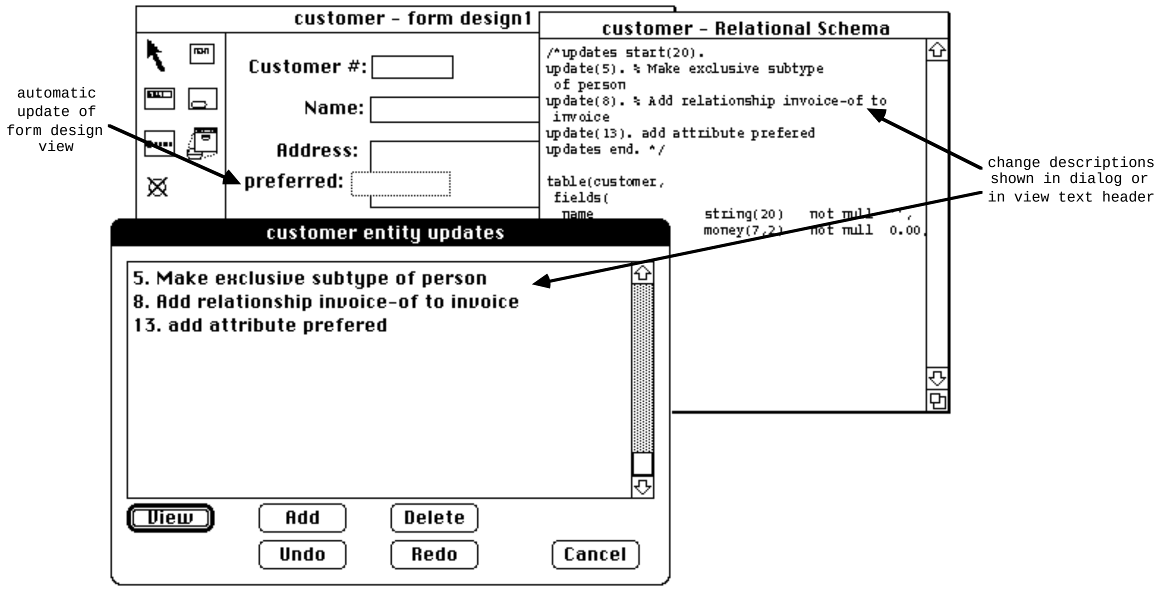
```
/*updates start(20).
update(5). % Make exclusive subtype
 of person
update(8). % Add relationship invoice-of to
 invoice
update(13). % add attribute prefered
updates end. */

table(customer,
 fields(
  name              string(20)   not null  '',
  credit limit      money(7,2)   not null  0.00,
```

```
/*updates start(4).
updates end. */

Item: entity
Name: customer
Relationships:
  acc-of, inv-of
Attributes:
  c#, name, address, age,
  credit limit
Purpose:

customer is used to record information
about business customers. It is used by
```

Figure 1. An integrated Information Systems Engineering Environment.

We have developed an ISEE which incorporates entity-relationship and relational schema modelling [7], form and report design [10], NIAM modelling [22], object-oriented analysis, design and implementation [6], and documentation [6, 7]. Figure 1 is a screen dump showing some of the IS development views this environment provides. Windows 'customer - ERD' and 'customer - Relational Schema' are provided by the MViewsER ER/schema modelling tool [7]. Window 'customer - form design1' is provided by the MViewsDP form/report design tool [10]. Window 'customer - Documentation' is a documentation view provided by the SPE tool for object-oriented software development (which also provides object-oriented analysis, design and implementation views) [6]. Other views supported include NIAM views from MViewsNIAM [22], and debugging views from CernoII [6].

Our integrated tools support integrated repositories, and information in these repositories and the tool views is kept consistent when related information in other views is modified. Figure 2 shows an example of consistency management. Updates made to the ER customer entity are propagated to the schema view, shown as *change descriptions* in this textual view header. These inform a developer of changes made to the customer entity that may require updates to the schema. Some can be automatically made by the environment (for example, adding, renaming or deleting attributes). Others are more difficult to implement, such as adding new relationships or changing relationship arity. The environment leaves these modifications up to the developer. Changes need to be reflected in the form design view, and these are indicted in a dialog. Some can be automatically actioned, as shown by the new attribute ‘preferred’ having an edit box automatically added, but this needs designer intervention to move it to an appropriate position (hence the greyed outline).


automatic update of form design view
customer - form design1
Customer #:
Name:
Address:
preferred:
customer - Relational Schema
change descriptions shown in dialog or in view text header
customer entity updates
View
Add
Delete
Undo
Redo
Cancel

```
/*updates start(20).
update(5). % Make exclusive subtype
 of person
update(8). % Add relationship invoice-of to
 invoice
update(13). add attribute prefered
updates end. */

table(customer,
 fields(
  name                 string(20)   not null   '',
                       money(7,2)   not null  0.00,
```

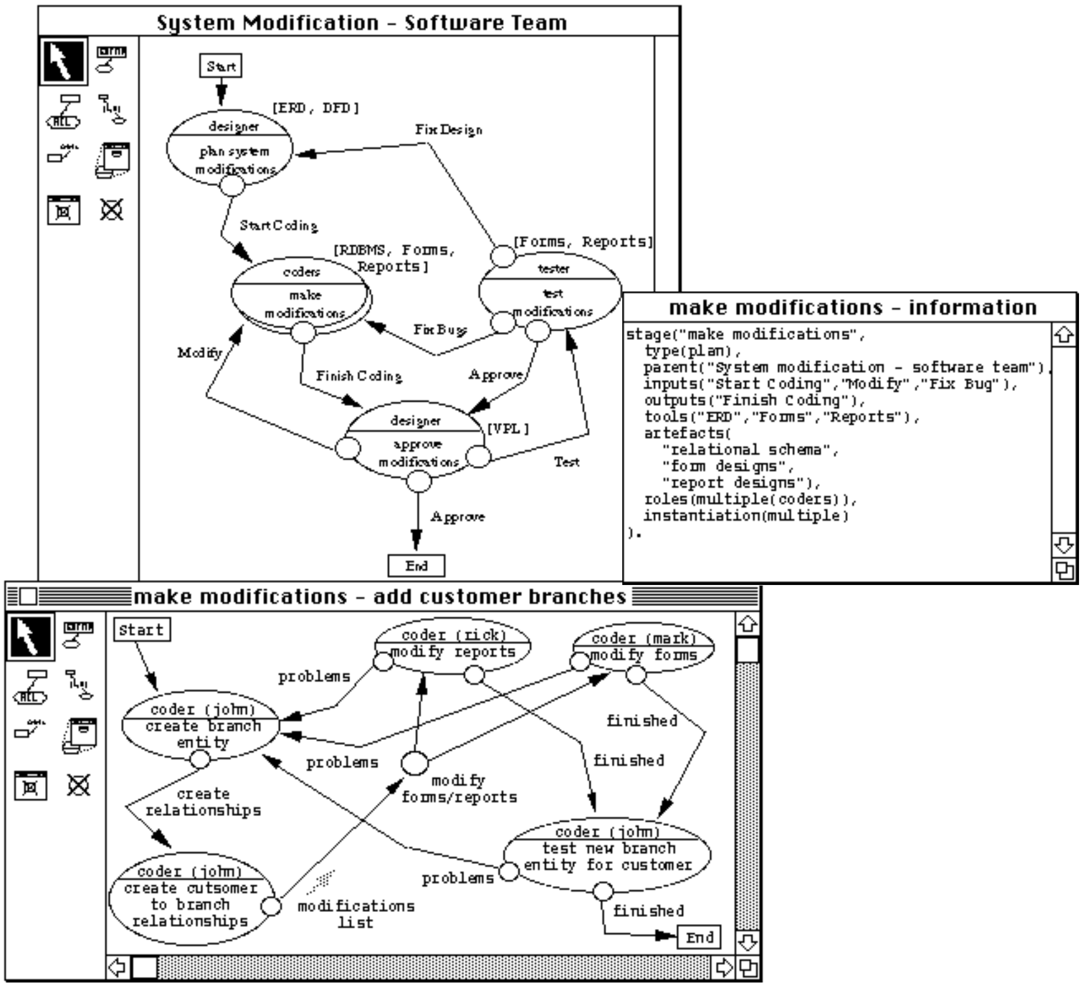
5. Make exclusive subtype of person
8. Add relationship invoice-of to invoice
13. add attribute prefered

Figure 2. Keeping different views of development consistent.

4. Supporting Process Modelling and Work Coordination

Our environment supports version merging and synchronous and semi-synchronous collaborative editing facilities [11].These are useful, but are not by themselves sufficient to support large groups of cooperating IS developers. A key problem is lack of information about work processes used and the context in which work artefact changes have been carried out [9]. In large cooperative work systems coordination of work activities is needed [14]. Users must collaborate in the planning of work activities and modelling of ISD processes, and be aware of the contexts in which other users’ work is carried out.

We have adapted the Visual Planning Language (VPL) [21] to define process models, work contexts and work planning activities for our ISEE. VPL defines plans and subplans for work, and can be used to specify meta-plans for planning itself. We have extended VPL to produce VPL+, in which plans include extra information about work artefacts, CASE tools and collaboration mechanisms. Two VPL+ plan diagrams for a software process are shown in figure 3. VPL+ elements include: stages (steps in the process), denoted by ellipses 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 stage perimeters, which specify the next plan stage(s). Extra annotations describe tools, work artefacts and communication mechanisms. These VPL+ elements (or "plan artefacts") can be modified in exactly the same ways work artefacts are modified. Plans can be defined by multiple users via synchronous, semi-synchronous or asynchronous editors. Modifications to plans are coordinated via meta-plans (which are themselves VPL+ plans)..



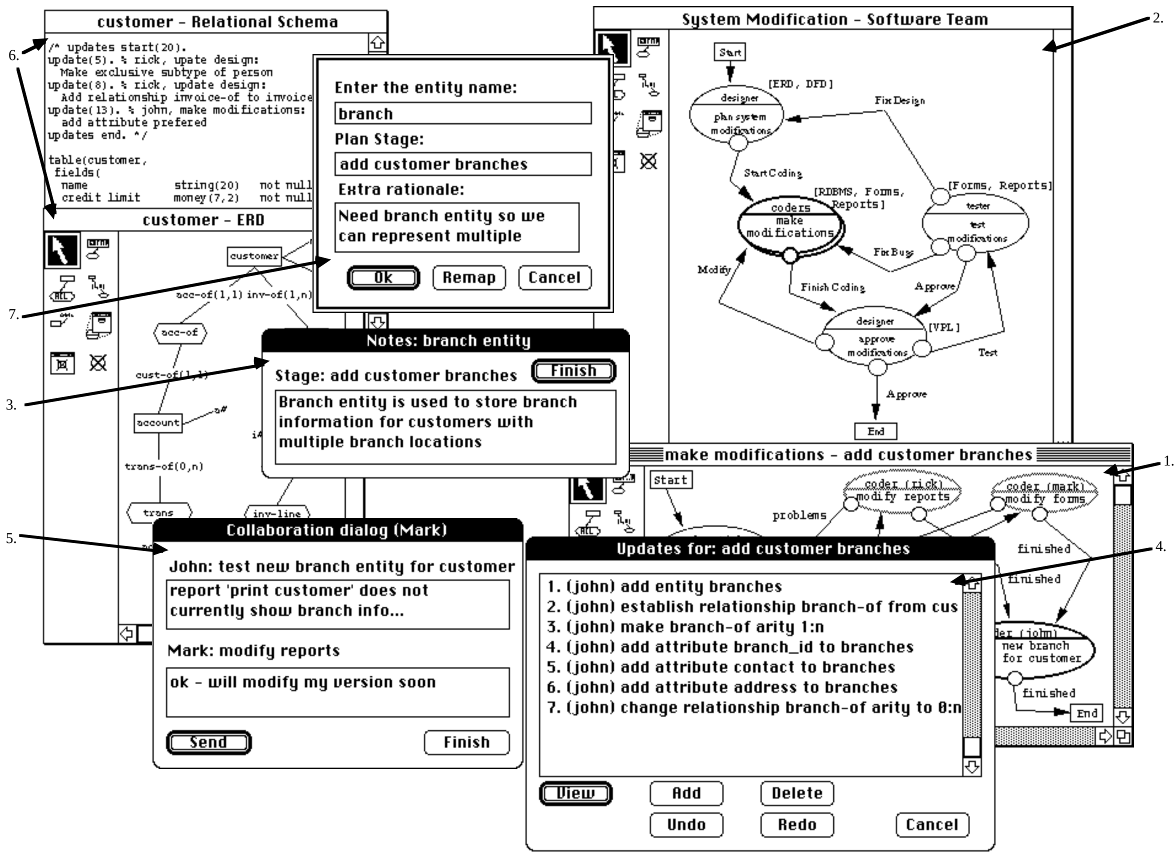
```
stage("make modifications",
  type(plan),
  parent("System modification - software team"),
  inputs("Start Coding","Modify","Fix Bug"),
  outputs("Finish Coding"),
  tools("ERD","Forms","Reports"),
  artefacts(
    "relational schema",
    "form designs",
    "report designs"),
  roles(multiple(coders)),
  instantiation(multiple)
).
```

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

In figure 3, the plan specifies a software process for modifying a software system. A subplan for “make modifications” 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 is associated with each actioned stage, containing descriptions of changes made to software artefacts while the stage was active. Textual views describe extra information about the purpose of individual plan stages. These describe the tools used to carry out work and which artefacts are used by a stage. Useful plans can be abstracted out into policies (basically generalised software process models), which can then be instantiated into further plans.

5. Capturing and Presenting the Context of Work

VPL+ is used to display and manipulate plans in action, with active stages for a plan highlighted. This enacted plan specifies the “current work” context for each member in a collaborating team. Members may have more than one plan view open to see the status of other plans they are interested in. 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 in which the change was made.



```
/* updates start(20).
update(5). % rick, upate design:
  Make exclusive subtype of person
update(8). % rick, update design:
  Add relationship invoice-of to invoice
update(13). % john, make modifications:
  add attribute prefered
updates end. */

table(customer,
 fields(
  name            string(20)   not null
  credit limit    money(7,2)   not null
```

Enter the entity name: branch

Plan Stage: add customer branches

Extra rationale: Need branch entity so we can represent multiple

Notes: branch entity

Stage: add customer branches

Branch entity is used to store branch information for customers with multiple branch locations

Collaboration dialog (Mark)

John: test new branch entity for customer

report 'print customer' does not currently show branch info...

Mark: modify reports

ok - will modify my version soon

Updates for: add customer branches

1. (john) add entity branches
2. (john) establish relationship branch-of from cus
3. (john) make branch-of arity 1:n
4. (john) add attribute branch_id to branches
5. (john) add attribute contact to branches
6. (john) add attribute address to branches
7. (john) change relationship branch-of arity to 0:n

Figure 4. Capturing work context information.

Figure 4 shows how different views can be used to capture work contexts when using ISEE views (a developer would not normally have this many views open at once!). A shared VPL+ view (1) shows information about a developer’s current active stage status (bold border) and indicates current plans of other collaborators (shaded borders). Other VPL+ views (2) show other plans the developer is using. Shared artefact notes (3) specify extra documentation about individual work or plan artefacts. Shared modification histories (4) detail changes that have been made for a stage. 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 work history. A collaboration dialog (5) is used for informal, context-dependent dialogue between developers. Work artefact views (6) are graphical or textual ISEE editors which capture information about changes made to artefacts. Augmented artefact dialogs (7) allow a developer to optionally specify extra rationale for low-level work or plan artefact changes. The current context of work is modified by advancing VPL+ active stages and VPL views can be extended as users become more familiar with their actual work processes, unlike many existing workflow and process-centred environments.

Collaborating users must be informed of changes to work and plan artefacts they are interested in [9]. 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 with both change descriptions describing actual work or plan artefact changes and extra information about the work context in which the changes were made. Presentation of work context information to collaborators is done in a similar way and using similar views to the capture of work context information shown in Figure 4 [9].

6. Method Engineering

Information Systems development methodologies are generally situation-independent. Researchers have found that due to the increasing complexity of Information Systems, methods often need to be tailored for a particular system development project[12]. Our work coordination views assist in Method Engineering by allowing developers to incrementally refine their development methodology and work processes. Our approach has advantages over comparable, textual notations, such as MEL [12], in that its visual nature is more accessible to developers for visualising and modifying plans. It also allows users to modify their processes during or after use, so new, improved policies (i.e. software process models) can be developed.

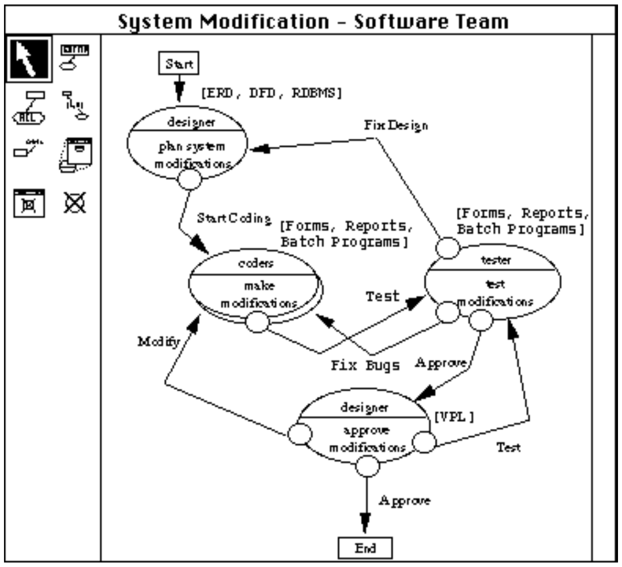


Figure 5. Method engineering techniques.

Figure 5 shows a VPL+ method engineering example. 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.

7. Design and Implementation

The individual ISEE tools and our VPL+ tool are implemented as a collection of classes, specialised from the MViews framework [5, 10]. MViews supports the construction of ISEEs 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 development tools. ISEE data is described 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. When a component is updated, a *change description* documenting the change is generated. Change descriptions are propagated to all components dependent upon the updated component's state. Dependent components interpret change descriptions and possibly modify their own state, producing further change descriptions. This mechanism supports a diverse range of environment facilities, including attribute recalculation, multiple views, flexible, bi-directional textual and graphical view consistency, component "modification histories" and versioning, and collaborative editing support [11]. New components and editing tools are constructed by reusing abstractions provided by the object-oriented MViews framework. A persistent object store is used to store component and view data.

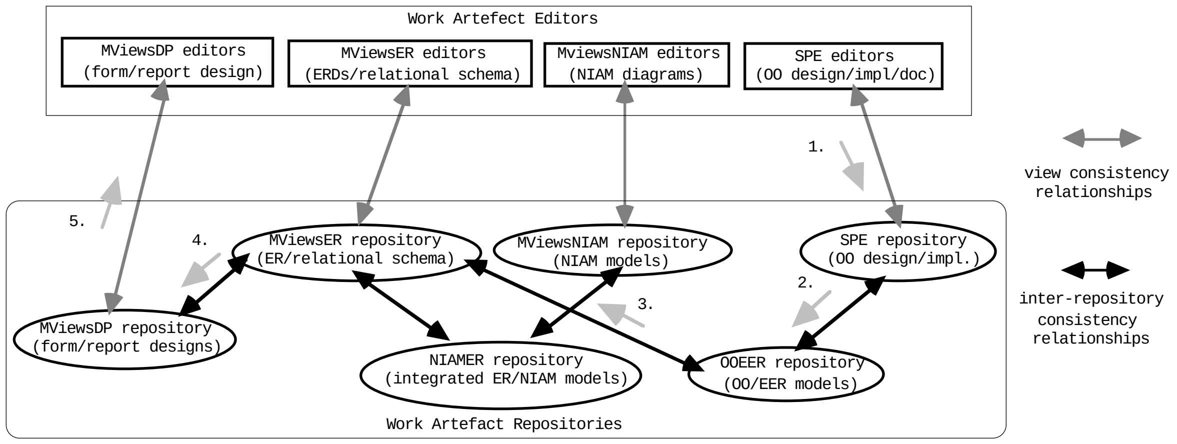

Work Artefect Editors
MViewsDP editors (form/report design)
MViewsER editors (ERDs/relational schema)
MviewsNIAM editors (NIAM diagrams)
SPE editors (OO design/impl/doc)
1.
2.
3.
4.
5.
view consistency relationships
inter-repository consistency relationships
MViewsER repository (ER/relational schema)
MViewsNIAM repository (NIAM models)
SPE repository (OO design/impl.)
MViewsDP repository (form/report designs)
NIAMER repository (integrated ER/NIAM models)
OOEER repository (OO/EER models)
Work Artefact Repositories

Figure 6. The architecture of the integrated tools.

We have developed a technique of integrating tools for multiple design notations using hierarchical, integrated MViews repositories [7]. We have 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). We have integrated the SPE OOA/D/P, MViewsER, MViewsNIAM, and MViewsDP tools to produce our integrated ISEE, as shown in Figure 6. 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 kept consistent using an integrated repository.

The approach we have taken 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 [9]. Figure 7 shows how ISEE and VPL+ artefacts will interrelate: 1) if a developer modifies a work artefact, change descriptions are sent to the current VPL+ plan stage (developer's work context) . 2) The plan stage augments the change descriptions with work context information, and forwards them to the work context of collaborators interested in the change. 3) Change descriptions are presented to collaborators in an appropriate manner to inform them of both work artefacts changes and collaborating users' work context, facilitating work coordination.

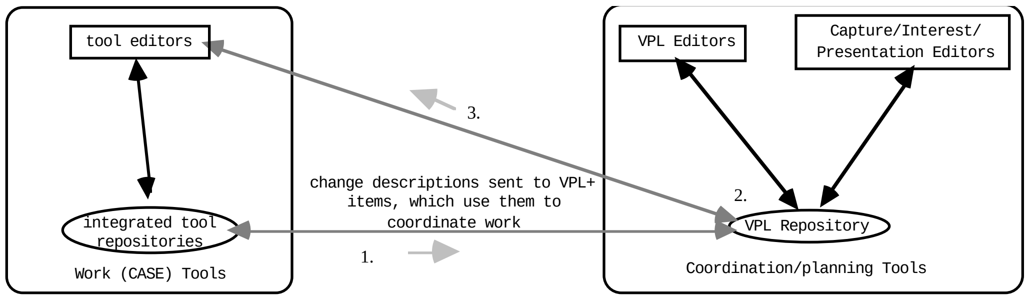

tool editors
integrated tool repositories
Work (CASE) Tools
3.
change descriptions sent to VPL+ items, which use them to coordinate work
1.
VPL Editors
Capture/Interest/ Presentation Editors
2.
VPL Repository
Coordination/planning Tools

Figure 7. Integrating VPL+ views with integrated tool views.

8. Summary

We have described an integrated ISEE 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 an extended Visual Planning Language (VPL+) is used to achieve process integration between tools. This includes the ability to collaboratively develop, refine and reuse complex work plans (i.e. software process specifications), coordinate work via work context capture and presentation, and use plans to document development via plan histories. 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 declaratively specified and generated.

References

1. Aaen, 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, North-Holland, Minneapolis, June 14-17 1992.
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,* Springer-Verlag, Finland, June 13-16 1995, pp. 269-282.
4. 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.
5. Grundy, J.C. and Hosking, J.G. A framework for building visusal programming environments. In *Proceedings of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, 1993, pp. 220-224.
6. Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B. Connecting the pieces,*Visual Object-Oriented Programming,* Manning/Prentice-Hall (1995), Chapter 11.
7. Grundy, J.C. and Venable, J.R. Providing Integrated Support for Multiple Development Notations. In *Proceedings of CAiSE'95,* LNCS 932, Springer-Verlag, Finland, Finland, June 1995, pp. 255-268.
8. 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,* 1995, pp. 109-116.

9. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D. Coordinating, capturing and presenting work contexts in CSCW systems. In *Proceedings of OZCHI'95*, Wollongong, Australia, Nov 28-30 1995, pp. 146-151.
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,* IEEE CS Press, April 5-7 1995, pp. 84-94.
11. Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Supporting flexible consistency management via discrete change description propagation. accepted for publication in *Software - Practice and Experience* (1995).
12. Harmsen, F., and Brinkkemper, S. Design and Implementation of a Method Base Management System for a Situational CASE Environment. In *Proceedings of the 2nd Asia-Pacific Software Engineering Conference (APSEC'95),* IEEE CS Press, Brisbane, December 1995.
13. Kaplan, S.M., Tolone, W.J., Bogia, D.P., and Bignoli, C. Flexible, Active Support for Collaborative Work with ConversationBuilder. In *1992 ACM Conference on Computer-Supported Cooperative Work*, ACM Press, 1992, pp. 378-385.
14. Krant, R.E. and Streeter, L.A. Coordination in Software Development.*CACM 38,* 3 (March 1995), 69-81.
15. 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.
16. 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.
17. 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.
18. Reiss, S.P. PECAN: Program Development Systems that Support Multiple Views. *IEEE Transactions on Software Engineering 11*, 3 (1985), 276-285.
19. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software 7,* 7 (July 1990), 57-66.
20. Reiss, S.P. Interacting with the Field environment. *Software practice and Experience 20*, S1 (June 1990), S1/89-S1/115.
21. Swenson, K.D. A Visual Language to Describe Collaborative Work. In *Proceedings of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, 1993, pp. 298-303.
22. Venable, J.R. and Grundy, J.C. Integrating and Supporting Entity Relationship and Object Role Models. In *to appear in Proceedings of the 14th Object-Oriented and Entity Relationship Modelling Conferece,* LNCS 1021, Springer-Verlag, Gold Coast, Australia, Dec 1995.
23. Wasserman, A.I. and Pircher, P.A. A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices 22*, 1 (January 1987), 131-142.