In Proceedings of SE:E&)'96, Jan 24-28 Dunedin, New Zealand, IEEE CS Press.

Serving up a Banquet: Towards an Environment Supporting All Aspects of Software Development

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

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

††Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand
{john, rick}@cs.auckland.ac.nz

Abstract

In order to make diverse software engineering techniques readily available and their use effective, they should be supported by appropriate CASE environments. We describe our experiences in building several such environments for different software engineering methodologies and notations. As these methodologies address different aspects of software development, tools embodying them should ideally be integrated within one overall environment. The design of such an environment is described, and our recent progress towards achieving this integrated environment for all aspects of software development is presented.

1. Introduction

Many diverse software engineering methodologies and notations are available for use by students, practitioners and researchers. To make effective use of these, environments and CASE tools supporting them are proliferating. Many techniques address different or overlapping aspects of the software development life cycle. For example, many programming environments utilise hierarchical browsers, with notations in common with those of CASE tools.

Tools supporting different aspects of software development can be used in isolation, but this results in much redundancy, an inability to effectively share data, and inconsistent user interfaces [21, 4]. Tools supporting different aspects of software development need to be integrated into a single Integrated Software Development Environment (ISDE) which overcomes these problems [21, 26, 24]. Many problems are presented when attempting tool integration. These include the need to: keep different views of software development consistent; effectively manage environment extension and evolution; and integrate tools shared by multiple users in multiple locations.

Recent research into integrating software development environments has focused on control, data, user interface and process integration [4, 21]. PECAN [25], MELD [18], and the Cornel Program Synthesiser [28] utilise a large centralised database to store shared information with structure editors allowing modification of views of 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 structure editing approach. CASE tools utilise code generation and reverse engineering to partially keep design and code consistent [33, 30]. Federated approaches use a database, such as PCTE, spread over several locations [4]. Tools supporting collaboration between multiple users include ConversationBuilder [19] and GroupKit [29]. Process-centred environments include CPCE [20] and Marvel [3]. None of these has satisfactorily solved all of the above problems.

We describe a heterogeneous approach to ISDE production. Tools supporting related aspects of software development are integrated via hierarchical repositories which support flexible, bi-directional consistency management. Within each tool, multiple textual and graphical views of the software artefact are supported and kept consistent. Multiple users are supported by a combination of low-level CSCW editing capabilities and high-level work and planning coordination facilities.

2. Towards Banquet

Our recent research has concentrated on building a variety of multi-view editing tools. These support :

- *object-oriented software development.* SPE (Snart Programming Environment) [9] is a multi-phase environment for OOA/D/P. Object-Z views have recently been integrated with SPE [11].
- *program visualisation.* CernoII [6] supports visualisation and debugging of OO programs.

- *user interface specification and construction.* MViewsDP [12] supports dialog box design. Skin [17] supports flexible user interface components. Currently under development are DrawByWire, a novel constraint-based user interface development tool, and LC+, which adds LeanCuisine+ [23] views to MViewsDP.
- *data modelling.* MViewsER and MViewsNIAM provide ER and NIAM modelling facilities [13, 32].
- *visual programming.* ViTABaL [14] supports visual tool abstraction based programming.
- *collaborative software development.* C-SPE provides collaborative extensions to SPE for multi-user OO development [15],
- *work coordination.* Under development is a high-level work and planning coordination tool [16].

Some of these tools have been integrated together loosely or tightly, while others are yet to be integrated in any way. Figure 1 illustrates these current "islands" of integration. For example, SPE and MViewsER have been fully integrated to produce OOEER, an environment which supports integrated OOA and EER modelling [13]. A similar process has been used to produce NIAMER, an environment supporting integrated ER and NIAM modelling [32]. Object-Z views have been fully integrated into SPE [11]. We are currently integrating the work coordination tool with C-SPE, and plan to extend this to support collaborative software development in our other tools [16].

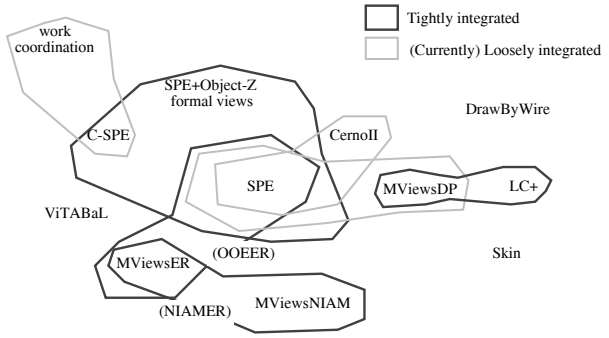

Figure 1. Current “islands” of integration.

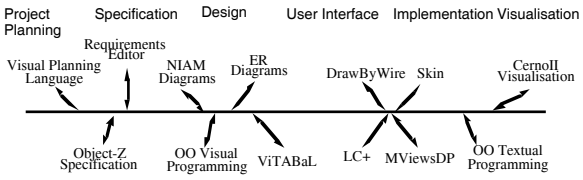

Figure 2. The Banquet integrated environment.

Our aim is to integrate all of the tools described above into, what appears to be from a user’s perspective, one ISDE called Banquet. Banquet will support all aspects of software development from planning and requirements formulation to program implementation, visualisation and documentation. Figure 2 illustrates the kinds of tools and the software life cycle tasks Banquet will support. In the next section we describe a framework for implementing tool integration. We then describe our existing tools in more detail, and our plans for integrating them into Banquet.

3. MViews

The tools we have already built are, in the main, implemented using the MViews architecture [8, 10]. MViews supports the construction of ISDEs by providing a general model for defining software system data structures and views, combined with a flexible mechanism for propagating changes between software components, views and tools.

ISDE 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 a base data dictionary graph. 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, as described in [10, 13].

When a software or view component is updated, a *change description* is generated. This is of the form `UpdateKind(UpdatedComponent, ... update-specific Values ...)`. For example, an attribute update on `Comp1` of attribute `Name` is represented as: `update(Comp1, Name, OldValue, NewValue)`.

All 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. See [8, 10, 15, 16] for details of the implementation of MViews.

MViews achieves control integration between tools via change description broadcasting [10]. Data integration is achieved via single or multiple hierarchical base views [13]. Presentation (user interface) integration is achieved through common user interface building blocks provided by the MViews framework [8]. Process integration is achieved at a low-level via collaborative editing tools [15], or via a high-level work coordination system [16].

MViews is implemented using Snart, an OO extension to Prolog. We are currently porting MViews to C++ to improve the availability, portability and efficiency of derived environments. This will provide Banquet's underlying integration mechanism.

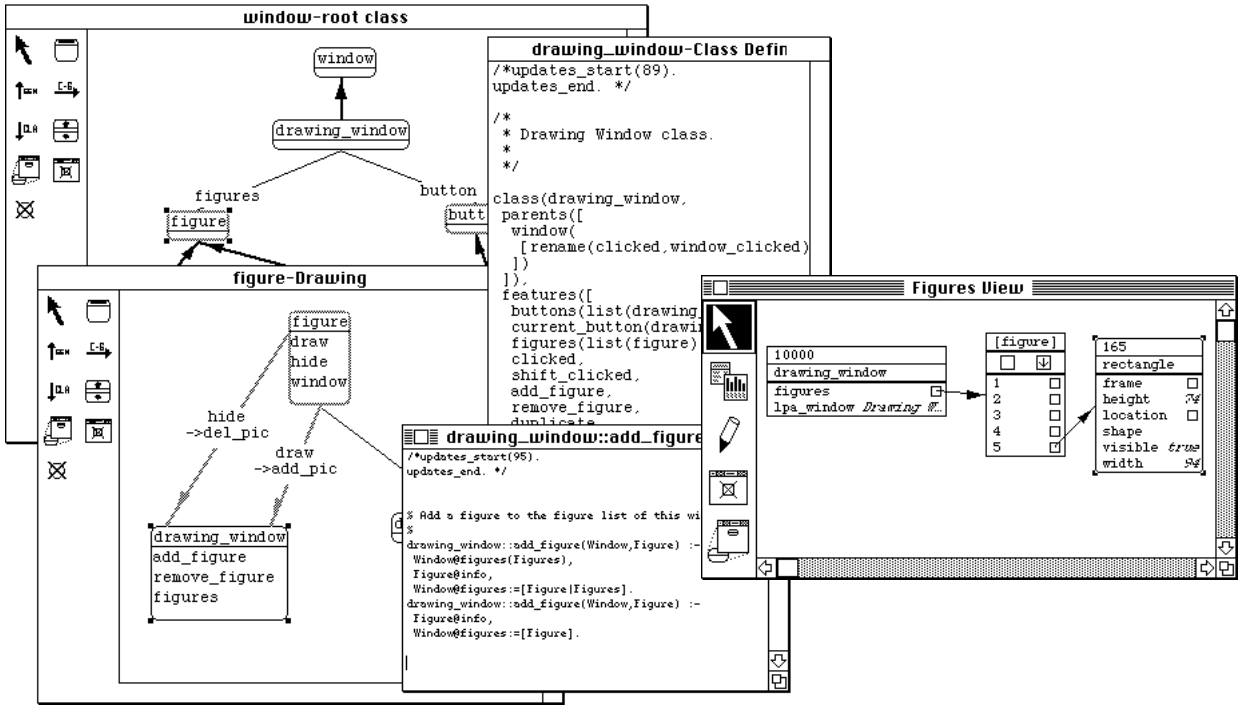

Figure 3. A screen dump from SPE.

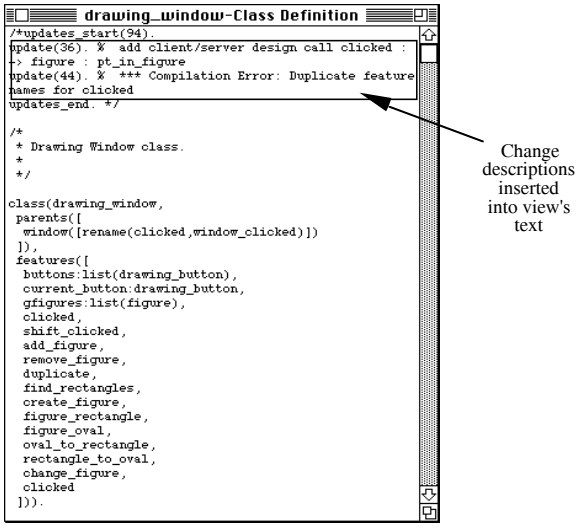

Figure 4. Textual view consistency.

4. ISDE for Object-oriented Development

We have used the existing MViews framework to build various tools for Analysis, Design, Implementation, and Visualisation. SPE is the major one, an ISDE which provides multiple textual and graphical views for constructing programs in Snart. SPE supports integrated analysis, design, implementation, and documentation tools.

Figure 3 shows a screen dump from SPE with two graphical views (one for analysis and one for design) and two textual views (a class interface and a method implementation). A CernoII graphical debugging view is also shown. Consistency is maintained between all SPE views, so changes to one view are always reflected in other views that share the updated information, no matter how loose the connection between the view representations. Additional analysis and design views, such as class contract and documentation views, are also provided by SPE. SPE supports a rich set of view navigation facilities, utilising hypertext techniques.

Graphical views are interactively edited and are kept consistent with other views by the environment directly updating changed icons. Descriptions of changes affecting graphical view components can also be viewed in dialog boxes. Textual views are free-edited and parsed. Textual view consistency involves expanding change descriptions into the view's text in a special header annotation, as shown in Figure 4. Some changes can be automatically applied by SPE to update the view's text, such as renaming classes and features and adding or deleting features. Other changes can not be carried out by the environment (eg a semantic error, or a design level change, such as the addition of a client-supplier relationship), and are manually implemented.

CernoII [6] provides graphical program visualisation views for running Snart programs. These include basic object attribute values, object reference networks and method call timing diagrams. CernoII and SPE are currently loosely integrated. Both can be running at the same time, and both have a common look-and-feel.

We have recently added Object-Z formal specification views to SPE [11]. Figure 5 shows a screen dump which illustrates how SPE design and implementation views can be kept consistent with Object-Z views. Change descriptions are expanded into the relevant class interface and method implementation views when an Object-Z view is modified, and vice-versa. Some of these, such as adding attributes and method arguments or renaming attributes and methods, can be automatically carried out by the environment. Others are documented by the environment for programmers to manually implement.

We are porting SPE and CernoII to C++, to support integrated software development of C++ programs. This requires extension of SPE's graphical notation to cope with additional constructs available in C++ and a modification of CernoII to interface to the C++ runtime system. Tighter integration between the tools will be provided via a facility to move between SPE and CernoII views via hyperlinks. These OO development tools will provide the core of the Banquet environment.

5. Other modelling tools

SPE provides facilities for object-oriented modelling. We have also developed tools for modelling using other approaches. MViewsER [13] provides graphical Entity-Relationship model views and textual relational schema views. These views are kept consistent in a similar manner to SPE graphical and textual views, and the textual views can be exported to a relational database system. MViewsNIAM provides similar facilities for NIAM modelling [32].

In Banquet, we wish to allow a variety of modelling approaches to be used in a single project, with translation and consistency maintained between those models that overlap in content. As a first step in such an integration, we have integrated SPE and MViewsER to produce OOEER, an environment supporting truly integrated OOA and EER modelling [13].

Figure 6 shows a screen dump from OOEER. OOA/D views are kept consistent with **all** changes to EER views, and vice-versa, even when direct translation is not possible. The dialog shown holds change descriptions (the "modification history") for the customer OOA class. Change descriptions highlighted by '→' were actually made to the EER view (diagram) and automatically translated into OOA/D view updates (where possible) by OOEER. Unhighlighted items were made by the designer to the OOA view to fully implement "indirect" translations that could only partially by implemented by OOEER.

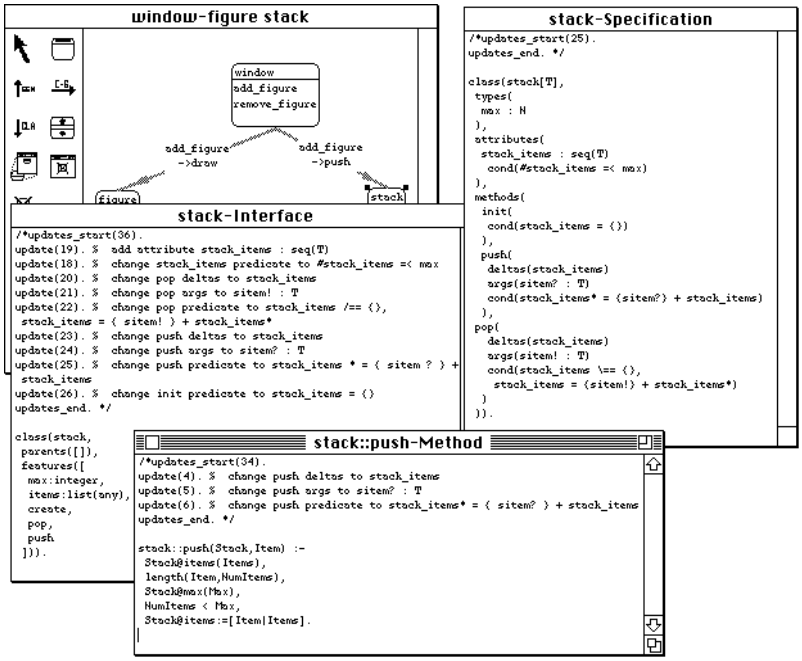

Figure 5. Integrated Object-Z views in SPE with bi-directional consistency management.

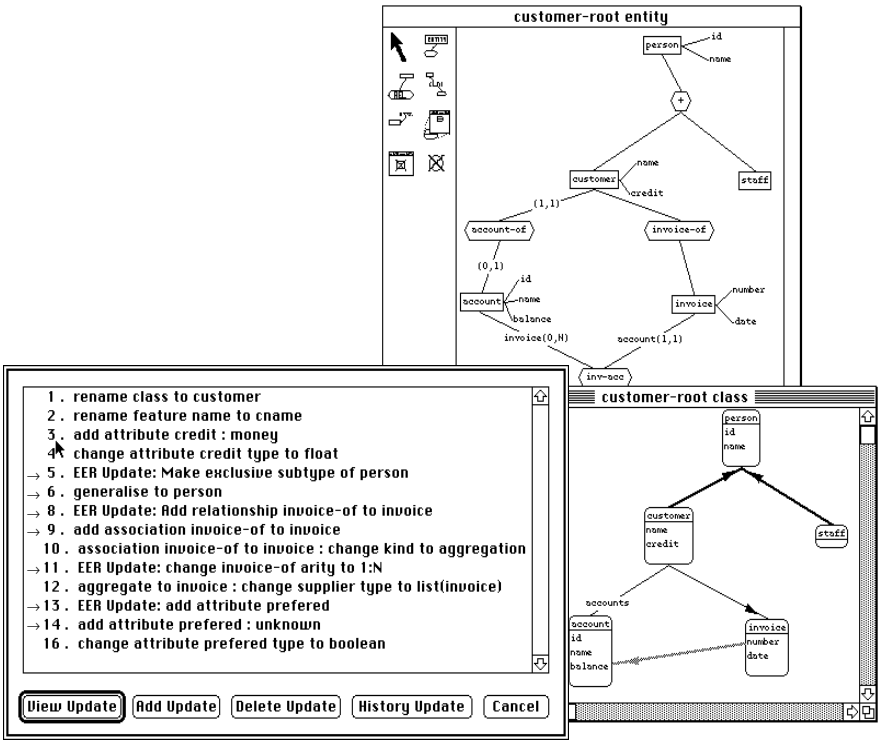

Figure 6. Integrated OOA/D and EER views in OOEER with bi-directional consistency management.

The OOEER integration was achieved by adding an additional data dictionary graph level below the data dictionaries of the SPE and MViewsER tools. This layer is responsible for translating, where possible, between the different notations and notifying tools where automatic translations are not possible. Neither SPE nor MViewsER required any significant change to achieve this integration.

In a similar manner we have produced NIAMER, a design environment integrating MViewsER and MViewsNIAM to produce an integrated NIAM/ER modelling environment [32].

The success of the OOEER and NIAMER integrations gives us good reason to believe that the MViews layered dictionary approach to integration will scale up for integrating the many tools in the Banquet project.

6. Visual Programming

The tool abstraction (TA) paradigm [7] is a new software engineering design and implementation approach. TA involves designing programs around functional units (called toolies) which share a pool of abstract data structures (ADSs). TA produces systems which can be more readably adapted to functional specification changes than traditional data abstraction.

A variety of approaches to implementing tool-abstraction based designs have been suggested. These include active data in OO systems, spreadsheets, structure-oriented editors, and rule-based systems [7]. We have recently developed ViTABaL (Visual Tool Abstraction Based Language) which provides a general-purpose, visual and textual specification language and environment for building TA-based systems [14]. ViTABaL makes tool abstraction more readily available for system designers and programmers.

Figure 7 shows a screen dump from ViTABaL showing an event propagation view describing the flow of events between toolies and their ADSs. This example describes a simple design for a KWIC (Key Word In Context) indexing system [22, 7], which generates a set of sorted, shifted words from input lines. Designers build up toolie and ADS structures and connect these via event connections, describing the kind of event flows between them. This includes "listening" to events before or after they have been received by toolies, and even modifying result data values on a listen-before [14].

ViTABaL supports the design of very flexible TA systems, allowing designers to easily change toolies from serial to concurrent operation, from batch to incremental processing, and to modify toolie or ADS behaviour via listening without changing existing code.

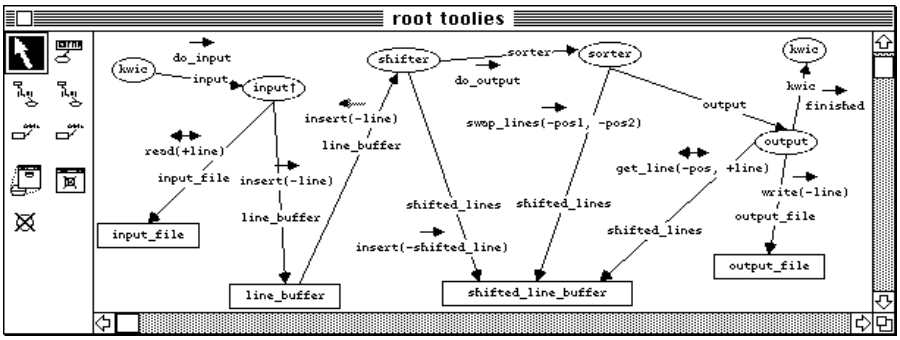

Figure 7. An example of a ViTABaL event propagation view.

ViTABaL uses textual views for specifying how toolies respond to events, using an adapted form of Snart syntax. Event propagation views can be animated to show the flow of events through running TA systems.

ViTABaL generates Snart code. This could invoke or be invoked by code developed in other tools, such as SPE and MViewsDP. As part of Banquet development, we are integrating ViTABaL into the ISDE to allow programs to be built using traditional object-oriented approaches, as supported by the SPE approach, or tool abstraction, as supported by ViTABaL.

7. User Interface Construction

We have built several tools with MViews and Snart for user interface construction. MViewsDP [12] provides graphical interface builder-style views for defining the appearance and location of user interface components, such as dialog boxes. Textual views provide additional information such as the interface to dialogs, dialog control parameters, and constraints on control values and behaviour. A dialog specification generates Snart code which is run from within the environment. We are currently adding a LeanCuisine+ [23] view to MViewsDP, which will complement its interactive specification and textual interface/constraint views. This will allow the structure of the user interface components to be more easily specified and visualised, and many constraints will be more abstractly specified by annotating this structure.

Another tool, arising from the CernoII visualisation work, is the Skin visual, functional language. Skin supports the definition of flexible user interface components which are highly adaptable to data change [17], in contrast to the more static dialog-box like structures able to be specified using MViewsDP.

Figure 8 shows a Skin function. This takes two arguments and constructs a horizontal list containing a textually formatted version of the first argument (using the text formatter primitive text), a block of white space (□), an alignment marker (see below) and a horizontal bar (▬), with length specified by the second argument's value. Alignment (⊢⊣)is used to line up icon components across multi dimensional lists. A horizontal list constructor (□□□)is used to gather the above elements together horizontally in sequence. Attached to the function result pin is a viewer (▭). Viewers render a prototype version of the Skin fragment they are attached to.

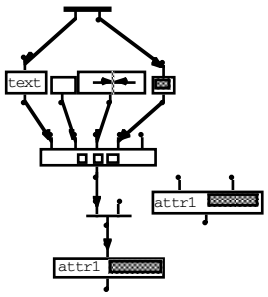

Figure 8. An example Skin specification.

MViewsDP and Skin can currently be used in conjunction with SPE. Code generated by these systems can be called from SPE Snart programs, and thus much of the user interface aspects of these programs can be defined in Skin and MViewsDP. There is not true integration between these environments with no hyperlinks between views nor shared data. Banquet will integrate these tools, allowing a range of user interface specification mechanisms to be used on a single project.

8. Integrating multiple users

We have made progress on integrating multiple users of an environment via both low-level editing facilities and high-level coordination facilities. C-SPE is a collaborative environment for software development [15]. Figure 9 shows a screen dump from C-SPE during semi-synchronous view editing. Changes made by a collaborating user “rick” are presented semi-synchronously in work artefact views or in a dialog. C-SPE also supports asynchronous development, where different versions are merged, and synchronous view editing, where users see and manipulate exactly the same view information. C-SPE was built by extending MViews to produce C-MViews, incoporating support for collaborative editing facilities.

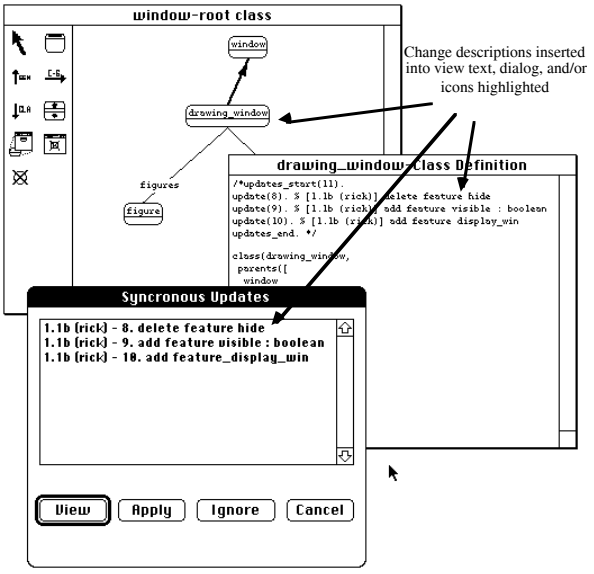

Figure 9. Semi-synchronous editing in C-SPE.

While C-SPE provides useful collaborative support for software development, C-SPE's low-level editing mechanisms do not support the coordination of work, and only deal with the editing of work artefacts. C-SPE thus illustrates a key problem for systems providing low-level support for CSCW work: a lack of information about the context that work artefact changes have been carried out in. The collaborating user is not told why rick carried out his changes, 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.

We have designed extensions to C-MViews which support more effective coordination of collaborative work [16]. Collaborating users design work plans together, and can abstract these into policies, using VPL (Visual Planning Language) [31]. Both plans and policies are very flexible and can be modified before, during or after actioning. Work artefact changes are carried out in association with a current plan stage, the current work context. Descriptions of work and plan artefact changes are augmented with extra information, which captures the current work or planning context, and any extra rationale for the changes. Collaborating users register their interest in plan, policy, or work artefact changes, so that when these artefacts are modified by collaborating users, the user is informed of the change in an appropriate manner. Collaborating users are informed of changes by the presentation of both the actual changes carried out and the context the work was carried out in. This includes grouping change descriptions with their plans or meta-plans, showing collaborators the context of work when displaying change descriptions, and highlighting affected plan and work artefacts in views in various ways.

9. Summary

We have described the Banquet Integrated Software Development Environment. Banquet will make a range of related software engineering notations and tools more accessible, and their use together more effective, for software practitioners, students and researchers. Systems developed using a variety of notations or methodologies will have all their views of software development kept consistent with one another. All data will be stored in a hierarchical data repository, and a consistent user interface amongst all tools will be provided. Collaborating developers will be able to coordinate their work more effectively than with current environments.

Current work Banquet is on three fronts. The first involves the transfer from the existing Snart-based implementation platform to C++. C++ offers significant performance and portability improvements over Snart. The second is the transfer to C++ as the target language for the ISDE. This is because the demand for a Snart ISDE is minimal in comparison to that for a C++ one. The third is a continuation of the integration of the environment islands described in Section 2. Hierarchical, integrated repositories are being used to achieve data integration between the tools. MViews' change propagation mechanism provides control integration. Presentation integration is via a consistent user interface, based on OpenDoc [2]. Process integration is by the use of our work coordination system, allowing users to plan and manage complex, cooperative work with many different tools and notations. The result will be an ISDE with powerful consistency management and collaborative mechanisms that supports: object-oriented analysis, formal specification, design, C++ implementation, and visualisation; flexible, high-level user interface specification and construction tools; visual programming for tool abstraction designs; and a high-level work coordination system.

References

[1] Apperley, M.D. and Spence, R., "Lean Cuisine: A low-fat notation for menus," *Interacting With Computers*, vol. 1, no. 1, 43-68, 1989.

[2] *OpenDoc Users Manual*, Apple Computers Inc.

[3] Barghouti, N.S., "Supporting Cooperation in the Marvel Process-Centred SDE," in *Procs of the 1992 ACM Symposium on Software Development Environments,* ACM Press, 1992, pp. 21-31.

[4] Bounab, M. and Godart, C., "A Federated Approach to Tool Integration," in *Procs of CAiSE'95,* Finland, June 1995, LNCS 932, Springer-Verlag, pp. 269-282.

[5] Cox, P.T., Giles, F.R., and Pietrzykowski, T., "Prograph: a step towards liberating programming from textual conditioning", in *Procs of the 1989 IEEE Workshop on Visual Languages,* 1989, IEEE CS Press, pp. 150-156.

[6] Fenwick, S., Hosking, J.G., and Mugridge, W.B., "Visual debugging of object-oriented systems," in *Procs of TOOLS Pacific 94,* 1994.

[7] Garlan, D., Kaiser, G.E., and Notkin, D., " Using Tool Abstraction to Compose Systems," *COMPUTER*, vol. 25, no. 6, 30-38, June 1992.

[8] Grundy, J.C. and Hosking, J.G., "A framework for building visusal programming environments," in *Procs of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, 1993, pp. 220-224.

[9] Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B., Connecting the pieces, Chapter 11 in *Visual Object-Oriented Programming*. Mnning/ Prentice-Hall, 1995.

[10] Grundy, J.C. and Hosking, J.G., "Constructing Integrated Software Development Environments with Dependency Graphs," Working Paper, Department of Computer Science, University of Waikato, 1994.

[11] Grundy, J.C., and Hosking, J.G., "Support for Integrated Formal Software Development," in *Procs of APSEC'95,* Brisbane, 1995, IEEE CS Press.

[12] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., "Supporting flexible consistency management via discrete change description propagation," Working Paper, Department of Computer Science, University of Waikato, 1995.

[13] Grundy, J.C. and Venable, J.R., "Providing Integrated Support for Multiple Development Notations," in *Procs of CAiSE'95,* Finland, June 1995, LNCS 932, Springer-Verlag, pp. 254-268.

[14] Grundy, J.C. and Hosking, J.G., "ViTABaL: A Visual Language Supporting Design By Tool Abstraction," in *Procs of the 1995 IEEE Symposium on Visual Languages,* IEEE CS Press, 1995.

[15] 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 1995, pp. 84-94.

[16] Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D., "Coordinating, capturing and presenting work contexts in CSCW systems," in *Procs of OZCHI'95,* Wollongong, November 1995.

[17] Hosking, J.G., Fenwick, S., Mugridge, W.B., and Grundy, J.C., "Cover yourself with Skin," in *Procs of OZCHI'95,* Wollongong, November 1995.

[18] Kaiser, G.E. and Garlan, D., "Melding Software Systems from Reusable Blocks," *IEEE Software*, vol. 4, no. 4, 17-24, July 1987.

[19] Kaplan, S.M., Tolone, W.J., Carroll, A.M., Bogia, D.P., and Bignoli, C., "Supporting Collaborative Software Development with ConversationBuilder," in *Procs of the 1992 ACM Symposium on Software Development Environments,* 1992, pp. 11-20.

[20] Lonchamp, J., "CPCE: A Kernel for Building Flexible Collaborative Process-Centred Environments," in *Procs of the 7th Conference on Software Engineering Environments,* Netherlands, April 1995, IEEE CS Press, pp. 95-105.

[21] Meyers, S., "Difficulties in Integrating Multiview Editing Environments," *IEEE Software*, vol. 8, no. 1, 49-57, January 1991.

[22] Parnas, D.L., "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, 1053-1058, December 1972.

[23] Phillips, C., "Serving Lean Cuisine+: Towards a Support Environment," in *Procs of OZCHI'94,* 1994, pp. 41-46.

[24] Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R., "Dora - a structure oriented environment generator," *IEE Software Engineering Journal*, vol. 7, no. 3, 184-190, 1992.

[25] Reiss, S.P., "PECAN: Program Development Systems that Support Multiple Views," *IEEE Transactions on Software Engineering*, vol. 11, no. 3, 276-285, 1985.

[26] Reiss, S.P., "Connecting Tools Using Message Passing in the Field Environment," *IEEE Software*, vol. 7, no. 7, 57-66, July 1990.

[27] Reiss, S.P., "Interacting with the Field environment," *Software practice and Experience*, vol. 20, no. S1, S1/89-S1/115, June 1990.

[28] Reps, T. and Teitelbaum, T., "Language Processing in Program Editors," *COMPUTER*, vol. 20, no. 11, 29-40, November 1987.

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

[30] *TurboCASE Reference Manual*, StructSoft Inc, 5416 156th Ave. S.E. Bellevue, WA, 1992.

[31] Swenson, K.D., "A Visual Language to Describe Collaborative Work," in *Procs of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, 1993, pp. 298-303.

[32] Venable, J.R. and Grundy, J.C., "Integrating and Supporting Entity Relationship and Object Role Models," in *Procs of the 14th Object-Oriented and Entity Relationship Modelling Conferece,* Gold Coast, December 1995, LNCS, Springer-Verlag.

[33] Wasserman, A.I. and Pircher, P.A., "A Graphical, Extensible, Integrated Environment for Software Development," *SIGPLAN Notices*, vol. 22, no. 1, 131-142, January 1987.