Coordinating Distributed Software Development Projects with Integrated Process Modelling and Enactment Environments

John Grundy[†], John Hosking[††] and Rick 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

Coordinating distributed software development projects becomes more difficult, as software becomes more complex, team sizes and organisational overheads increase, and software components are sourced from disparate places. We describe the development of a range of software tools to support coordination of such projects. Techniques we use include asynchronous and semi-synchronous editing, software process modelling and enactment, developer-specified coordination agents, and component-based tool integration.

Keywords: work coordination, distributed software development tools, process modelling, process-centred environments, computer-supported cooperative work

1. Introduction

Coordinating multiple developers working on a distributed software development project is very difficult, and gives rise to the following management problems:

- Developers need specific tasks assigned, which must be *coordinated* to ensure a working system results.
- Developers need to, at times, *communicate* and *collaborate* closely, while at other times can independently work on parts of a project.
- Software artefacts (code, designs, documentation etc.) need to be *shared* and kept *consistent*.
- Multiple tools must be used to modify artefacts, with some tools supporting close *collaborative editing* (e.g. via synchronous editing), while others supporting looser collaboration (e.g. via alternate version editing and subsequent merging).
- Progress towards specified goals needs to be tracked, developers need to remain *aware* of others' work, and complex software systems need to be configured from the constituent, distributively developed parts.
- Developers need to flexibly *configure* their environments' support for artefact management, communication, and work coordination.

Many systems have been developed which attempt to address these issues. Computer Supported Collaborative Work (CSCW) systems have been used to aid distributed software development. These include ConversationBuilder [16], wOrlds [6], Orbit [17], TeamRooms [24], Lotus Notes [19], and BSCW [5]. Programming environments, such as Mjølner [20], Mercury [15], and FIELD [24], may also provide basic collaborative software development facilities. However, while such systems may support shared editing and artefact management, they generally lack adequate coordination support.

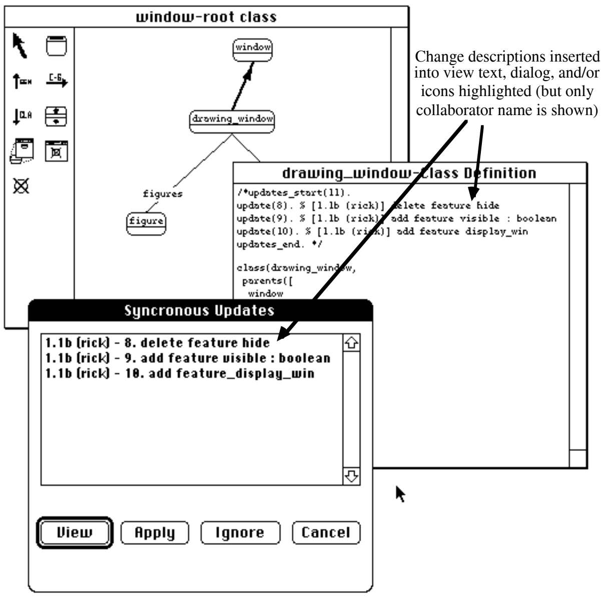
Process-centred Environments (PCEs), such as Oz [3], SPADE [1], ProcessWEAVER [7], and ADELE-TEMPO [2], more tightly integrate software development and software process support. Most such systems, however, provide complex mechanisms for specifying processes, have a limited range of work coordination strategies, and can be difficult to integrate with third-party tools. CSCW and process-centred environments can be usefully integrated. Examples include Oz [4], and SPADE-ImagineDesk [1], and some programming environments and process-centred environments, such as Multiview-Merlin [21]. Workflow and project planning systems, such as TeamFLOW [27], Regatta [26], CoMo-Kit [22], and Action Workflow [23], provide more accessible facilities for modelling work processes but generally lack flexibility for specifying work coordination mechanisms and tool integration.

In the following sections we describe our recent work addressing these problems of distributed software development coordination. Our solutions include annotating changes made to software artefacts and distributed to multiple users, tightly integrated software development and process modelling and enactment tools, and component-based software development, process modelling, and collaborative editing tools.

2. C-SPE

Our first attempt at supporting distributed software development was C-SPE [8]. SPE (Snart Programming Environment) is an integrated development environment for object-oriented software development using Snart, an Object-oriented Prolog [9]. SPE provides integrated OOA, OOD and Snart code views, along with debugging and documentation views. All are kept consistent under change by propagating representations of changes ("change descriptions") between views [9, 10].

C-SPE (Collaborative SPE) adds semi-synchronous and asynchronous editing to SPE. Distributed developers can simultaneously edit SPE views and be informed of changes other developers are making semi-synchronously. Developers distributed over time and space can edit different versions of software views asynchronously, and later merge the changes together. These modes of collaborative development are complementary; developers can move freely between them. A client-server architecture supports broadcast of editing changes to support semi-synchronous editing and provides a repository for storing versions to support asynchronous editing.


window-root class
window
drawing_window
figures
figure
drawing_window-Class Definition
Change descriptions inserted into view text, dialog, and/or icons highlighted (but only collaborator name is shown)
Syncronous Updates
View
Apply
Ignore
Cancel

```
/*updates_start(11).
update(8). % [1.1b (rick)] delete feature hide
update(9). % [1.1b (rick)] add feature visible : boolean
update(10). % [1.1b (rick)] add feature display_win
updates_end. */

class(drawing_window,
 parents([
  window
```

```
1.1b (rick) - 8. delete feature hide
1.1b (rick) - 9. add feature visible : boolean
1.1b (rick) - 10. add feature_display_win
```

Figure 1. Semi-synchronous editing in C-SPE.

Figure 1 shows an example of C-SPE being used by two developers, John and Rick, to semi-synchronously edit two views of a simple drawing editor program. The screen dump, from John's perspective, shows an OOD diagram and a Snart code view. The changes shown in the dialogue box have been made by Rick to his version of the OOD diagram. John is informed of these as they are made. Likewise, the change annotations at the top of the text view appear as Rick makes modifications that affect the view (changing the text and reparsing it or modifying shared information in the OOD view). The changes are annotated with Rick's name and the version number of the view(s) he is modifying (in this case alternate 1.1b of the OOD view). Asynchronous editing is also supported. Users may make changes to a view, then share the resulting list of changes with another developer for merging into their version of the view.

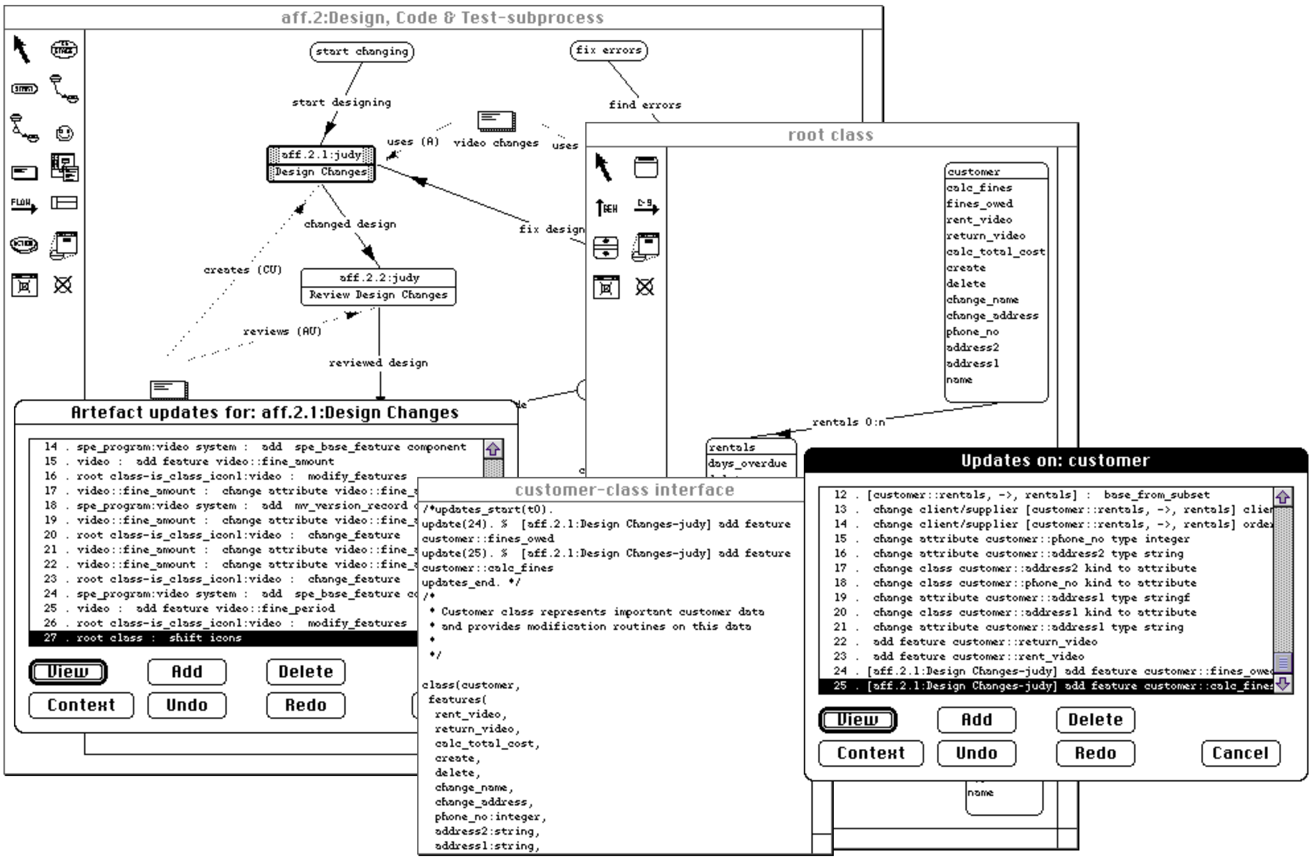
C-SPE suffers from a number of problems when deployed on distributed software projects. There is no "reason" why changes are made associated with change descriptions, nor any way to group changes to different artefacts according to what tasks different developers are performing. It is difficult for developers to determine why a specific change to an artefact was made by another developer, and difficult to determine what other artefacts were also affected to achieve a larger goal.

3. SPE-Serendipity

To address these problems, we developed Serendipity, a process modelling and enactment environment [13]. Serendipity provides an expressive, visual language for describing software processes and allows these processes to be enacted. We integrated C-SPE and Serendipity using their underlying message-passing architectures, resulting in an environment for distributed software development supporting integrated software development tools and process modelling and work coordination capabilities.

Figure 2 shows a screen dump from SPE-Serendipity in use during a software project. The contents of each window is a “view” into part of the software development project. Some views share information, others describe quite different aspects of the project. The top, left window shows an enacted software process in Serendipity (based on the ISPW6 software process example for system maintenance [18]). The rounded rectangles are "stages" in the software process, which have a unique ID, role (performer of the work), and name. Stages describe some subtask of the overall task of producing software, and may be further broken down into smaller subtasks. Stages are connected by labelled "enactment event" flows. When a stage is completed, enactment events flow to the connected stage(s), enacting them. Serendipity supports the modelling of process stages hierarchically, and the specification of artefacts, tools and roles used to perform these stages. Artefacts are created, used or updated by process model stages. Tools are used to access and/or modify artefacts. Roles indicate individuals, groups or abstract roles relevant to process model stages. The usage of different artefacts, tools and roles by process stages is indicated by “usage” connections between these icons.

Serendipity supports the coordination of work in SPE in various ways. Developers share Serendipity views, so can collaboratively plan and coordinate work on a project, and the refinement of software process models, at a high level of abstraction. They are also kept aware of the enacted stages of other developers by use of colouring and shading annotations of stage icons and enactment flows.



Artefact updates for: aff.2.1:Design Changes

```
14 . spe_program:video system :  add  spe_base_feature component
15 . video :  add feature video::fine_amount
16 . root class-is_class_icon1:video :  modify_features
17 . video::fine_amount :  change attribute video::fine_
18 . spe_program:video system :  add  mv_version_record c
19 . video::fine_amount :  change attribute video::fine_
20 . root class-is_class_icon1:video :  change_feature
21 . video::fine_amount :  change attribute video::fine_
22 . video::fine_amount :  change attribute video::fine_
23 . root class-is_class_icon1:video :  change_feature
24 . spe_program:video system :  add  spe_base_feature co
25 . video :  add feature video::fine_period
26 . root class-is_class_icon1:video :  modify_features
27 . root class :  shift icons
```

customer-class interface

```
/*updates_start(t0).
update(24). %  [aff.2.1:Design Changes-judy] add feature
customer::fines_owed
update(25). %  [aff.2.1:Design Changes-judy] add feature
customer::calc_fines
updates_end. */
/*
 * Customer class represents important customer data
 * and provides modification routines on this data
 *
 */

class(customer,
 features(
  rent_video,
  return_video,
  calc_total_cost,
  create,
  delete,
  change_name,
  change_address,
  phone_no:integer,
  address2:string,
  address1:string,
```

Updates on: customer

```
12 . [customer::rentals, ->, rentals] :  base_from_subset
13 .  change client/supplier [customer::rentals, ->, rentals] clien
14 .  change client/supplier [customer::rentals, ->, rentals] order
15 .  change attribute customer::phone_no type integer
16 .  change attribute customer::address2 type string
17 .  change class customer::address2 kind to attribute
18 .  change class customer::phone_no kind to attribute
19 .  change attribute customer::address1 type stringf
20 .  change class customer::address1 kind to attribute
21 .  change attribute customer::address1 type string
22 .  add feature customer::return_video
23 .  add feature customer::rent_video
24 . [aff.2.1:Design Changes-judy] add feature customer::fines_owed
25 . [aff.2.1:Design Changes-judy] add feature customer::calc_fines
```

Figure 2. Example of the SPE-Serendipity integrated environment in use.

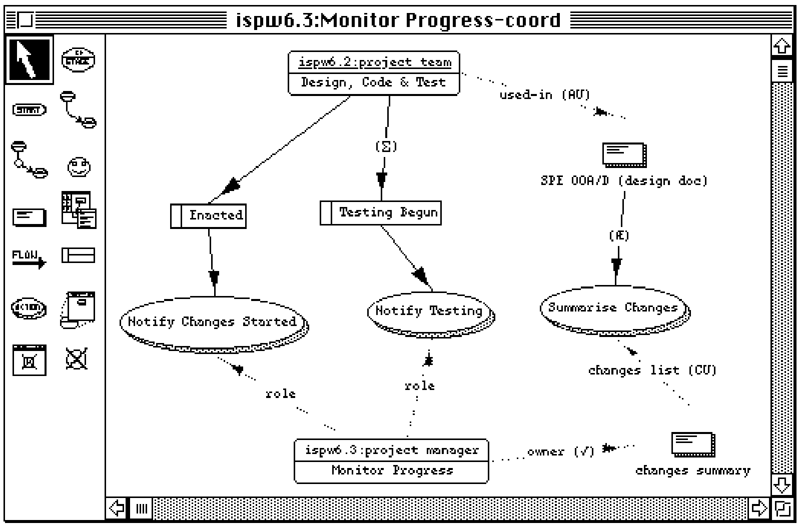


Figure 3. Simple work coordination specification.

Developers can also review changes made during stage enactment, and have access to shared, annotated histories of view and artefact changes made in SPE. These histories of work are automatically constructed by SPE and Serendipity. For example, the left hand dialogue in Figure 2 shows a history of modifications made while the “aff2.1:Design Changes" stage was the "current enacted stage" for a developer. This serves as a basic work history record, partitioned by stage. Developers share this history and thus can see what work others have performed on the project. The right hand dialogue in Fig. 2 shows the modification history of an SPE artefact (a class). The annotated change descriptions show the stage in which the changes were made, and are also shown in the centre, textual SPE code view of the class.

Serendipity also incorporates a complementary visual language which allows developers to configure the way they are informed of changes to artefacts of interest, the way work coordination is supported, and to specify automatic actions triggered by interesting events (artefact changes, or tool or process enactment events). Figure 3 shows an example of this "filters and actions" language used for coordinating a software development project. In this example, the project leader is informed when code modifications or testing begins (via an email message, broadcast message, dialogue or whatever is appropriate). In addition, all changes made to OOA/D artefacts in SPE are summarised and stored for the project leader to review at a later date. We have applied our filter and action models to many work coordination tasks for software development [13]. The visual nature of the language allows developers to build, reuse and modify these event-driven work coordination schemes more readily than the approaches used in most workflow and process modelling systems.

It is, however difficult to integrate third-party tools into Serendipity, and environment performance is poor, as it is implemented in Snart and only runs on Macintosh computers. Due to these performance problems, we have subsequently rearchitected our environment, as well as making some capabilities, such as the filters and actions language, useable in all of our development tools.

4. JComposer and Serendipity-II

Our earlier environments such as Serendipity and SPE were built using the Snart-based MViews framework, which supports the development of integrated software development environments [11]. We have subsequently reimplemented MViews, using Java, to produce the JViews framework. JViews is more platform-independent, provides tools with better performance, and allows third-party tools to be more easily integrated with our systems through Java Beans component technology [12]. We have also developed the JComposer tool for visually specifying and generating JViews-based environments [12]. JComposer generates Java Beans-based components which implement software development tools, and provides reusable components with C-SPE style collaborative editing, tool repository management, multiple view consistency mechanisms, and user interface components.

JComposer includes a form of the Serendipity filter and action language to specify event handling models for JViews tools. These filters and actions can be specified at run-time for these tools. This enables any JViews-based tool to support the degree of flexible, developer-controlled work coordination and user configuration Serendipity's filter and action language affords.

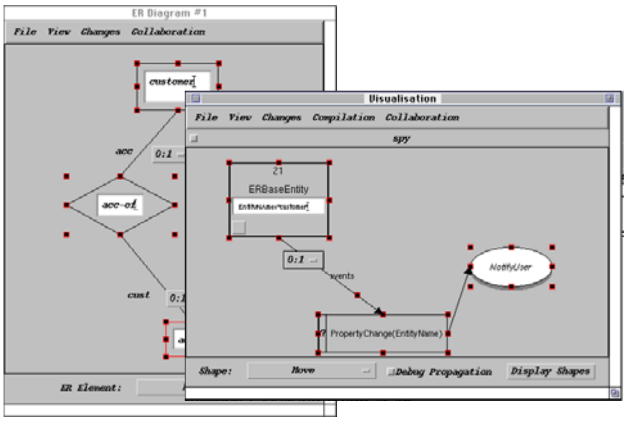


Figure 4. End-user specification of work coordination

Figure 4 shows a simple example of a JComposer-generated tool, a collaborative ER modeller. The right-hand view shows a JViews component which is being visualised (the "customer" entity), and a filter and action which have been added by the developer. Whenever the customer entity is renamed, the developer will be notified. The action component can be configured via a dialogue to notify the developer via an email message, broadcast message, highlighted icon in a view or via a dialogue box. Other work coordination schemes can be added at run-time by developers, for example constraining which artefacts and views can/can't be changed, automatically changing artefacts, or initiating dialogue with a developer when a process stage is enacted or artefact changed. [12].

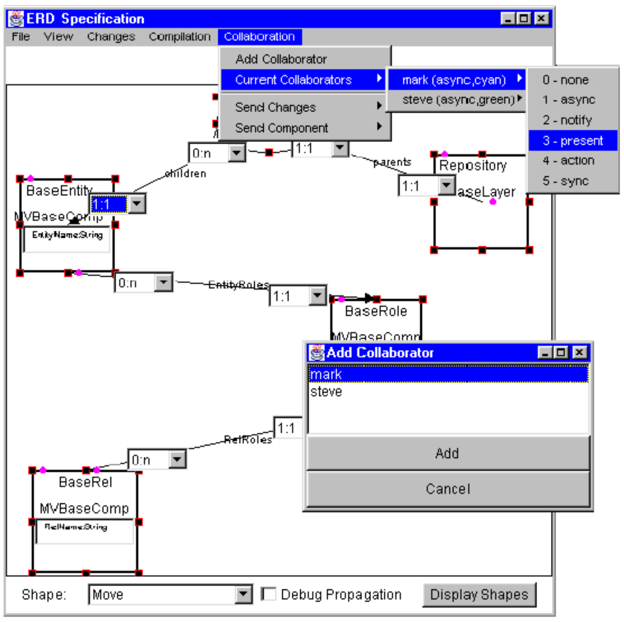
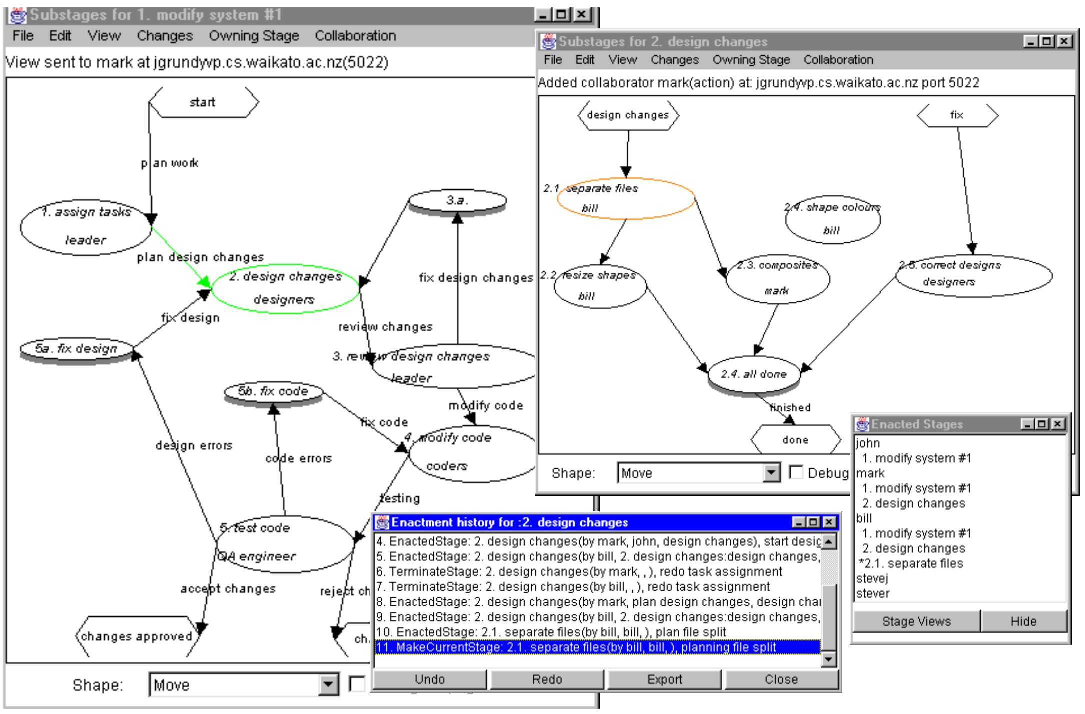


Fig. 5. Collaborative editing in JComposer.



Enactment history for :2. design changes

```
4. EnactedStage: 2. design changes(by mark, john, design changes), start desic
5. EnactedStage: 2. design changes(by bill, 2. design changes:design changes,
6. TerminateStage: 2. design changes(by mark, , ), redo task assignment
7. TerminateStage: 2. design changes(by bill, , ), redo task assignment
8. EnactedStage: 2. design changes(by mark, plan design changes, design chai
9. EnactedStage: 2. design changes(by bill, 2. design changes:design changes,
10. EnactedStage: 2.1. separate files(by bill, bill, ), plan file split
11. MakeCurrentStage: 2.1. separate files(by bill, bill, ), planning file split
```

Enacted Stages

```
john
  1. modify system #1
mark
  1. modify system #1
  2. design changes
bill
  1. modify system #1
  2. design changes
 *2.1. separate files
stevej
stever
```

Figure 6. The Serendipity-II process modelling environment showing an enacted software process.

Figure 5 shows JComposer in use, specifying the ER modeller repository from Figure 4. This development is being carried out collaboratively, with C-SPE style collaborative editing supported. Our JViews-based environments like JComposer and Serendipity-II support a decentralised form of user-configurable collaborative editing. In Figure 5 user "John" is configuring the "level" of collaborative editing with user "Mark" using a "collaboration menu". This is a component which has been plugged into the environment to provide a range of asynchronous to synchronous editing capabilities [14].

JComposer has been used to specify and generate a new, component-based version of Serendipity, Serendipity-II. This can be used with JComposer-generated tools to coordinate their use as with SPE-Serendipity. It can also use Java Beans components, interfaced via filters and actions, to be informed of third party tool events and to send instructions to such tools. This provides a very open architecture for coordinating the use of JViews-based and third-party distributed software development tools. Figure 6 shows Serendipity-II in use for coordinating work on a software development project. The right-hand dialogue shows other users and their enacted software process stages, and the left hand dialogue the enactment history of a process stage.

JComposer filters and actions are used in Serendipity-II to specify various work coordination schemes and to handle the automatic processing of events. This allows local and distributed software agents to be constructed by users of Serendipity-II to automate various tasks. Component-based tools which have been integrated with Serendipity-II, including all other JViews-based tools, can be interfaced to using appropriate filters and actions. Other tools can be interfaced to using file translation-supporting actions, or by building custom filters and actions specific to the data and control requirements of these tools.

Both JComposer and Serendipity-II are being used on a 7 person distributed software development project, in addition to other development tools, to assess their usability and functionality. To date both have performed well, and we expect to develop new filters and actions to integrate further tools, and develop new multiple view, multiple user software engineering tools with JComposer.

5. Summary

Distributed software development requires facilities to support collaboration, communication and coordination among software developers. We have developed a variety of solutions to these needs, including collaborative artefact editing mechanisms, and software process modelling tools to support coordination of this distributed work. Our approaches use component-based architectures, allowing our tools to have such facilities added to them rather than having to be built-in, and to be more readily observed and controlled by our process modelling environment.

We are currently enhancing the process modelling languages and software architecture of Serendipity-II to improve its performance for coordinating distributed software development. This includes the provision of integrated communication tools, the management of enactment and artefact change events via a database, and the development of interfaces to diverse third-party tools.

References

[1] Bandinelli, S., DiNitto, E., and Fuggetta, A., "Supporting cooperation in the SPADE-1 environment," *IEEE Transactions on Software Engineering*, vol. 22, no. 12.

[2] Belkhatir, N., Estublier, J., and Melo, W.L., *The Adele/Tempo Experience*, Software Process Modelling & Technology. Research Studies Press, 1994.

[3] Ben-Shaul, I.Z., Heineman, G.T., Popovich, S.S., Skopp, P.D. amd Tong, A.Z., and Valetto, G., "Integrating Groupware and Process Technologies in the Oz Environment," in *9th International Software Process Workshop,* Ghezzi, C., IEEE CS Press, Airlie, VA, October 1994, pp. 114-116.

[4] Ben-Shaul, I.Z. and Kaiser, G.E., "Integrating Groupware Activities into Workflow Management Systems," in *7th Israeli Conference on Computer Based Systems and Software Engineering,* Tel Aviv, Israel, June 1996, pp. 140-149.

[5] Bentley, R., Horstmann, T., Sikkel, K., , and Trevor, J., "Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system," in *In Proceedings of the 4th International WWW Conference,* Boston, MA, December 1995.

[6] Bogia, D.P. and Kaplan, S.M., "Flexibility and Control for Dynamic Workflows in the wOrlds Environment," in *Proceedings of the Conference on Organisational Computing Systems,* ACM Press, Milpitas, CA, November 1995.

[7] Fernström, C., "ProcessWEAVER: Adding process support to UNIX," in *2nd International Conference on the Software Process,* IEEE CS Press, Berlin, Germany, February 1993, pp. 12-26.

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

[9] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., "Supporting flexible consistency management via discrete change description propagation," *Software - Practice & Experience*, vol. 26, no. 9, 1053-1083, September 1996.

[10] Grundy, J.C., Venable, J.R., Hosking, J.G., and Mugridge, W.B., "Coordinating collaborative work in an integrated Information Systems engineering environment," in *Proceedings of the 7th Workshop on the Next Generation of CASE tools,* Crete, 20-21 May 1996.

[11] Grundy, J.C. and Hosking, J.G., "Constructing Integrated Software Development Environments with MViews," *International Journal of Applied Software Technology*, vol. 2, no. 3-4, 1996.

[12] Grundy, J.C., Mugridge, W.B., and Hosking, J.G., "A Java-based toolkit for the construction of multi-view editing systems," in *Proceedings of the Second Component Users Conference,* Munich, Germany, July 14-18 1997.

[13] Grundy, J.C. and Hosking, J.G., "Serendipity: integrated environment support for process modelling, enactment and work coordination," *Automated Software Engineering*, vol. 5, no. 1, 1998.

[14] Grundy, J.C., "Engineering component-based, user-configurable collaborative editing systems," Working Paper, Dept. of Computer Science, University of Waikato, 1998.

[15] Kaiser, G.E., Kaplan, S.M., and Micallef, J., "Multiuser, Distributed Language-Based Environments," *IEEE Software*, vol. 4, no. 11, 58-67, November 1987.

[16] 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.

[17] Kaplan, S.M., Fitzpatrick, G., Mansfield, T., and Tolone, W.J., "Shooting into Orbit," in *Proceedings of Oz-CSCW'96,* University of Queensland, Brisbane, Australia, August 1996.

[18] Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., and Rombach, H.D., "Software Process Modelling Example Problem," in *Proceedings of the 6th International Software Process Workshop,* (Ed), T.K., IEEE CS Press, Hokkaido, Japan, 28-31 October 1990.

[19] Lotus Inc., *System Administration Manual*, Lotus Notes release 3, 1993.

[20] 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.

[21] Marlin, C., Peuschel, B., McCarthy, M., and Harvey, J., "MultiView-Merlin: An Experiment in Tool Integration," in *Proceedings of the 6th Conference on Software Engineering Environments,* IEEE CS Press, 1993.

[22] Maurer, F. Project Coordination in Design Processes, In Proceedings of WET ICE'96, June 19-21, Stanford University, USA, IEEE CS Press.

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

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

[25] Roseman, M. and Greenberg, S., "A Tour of Teamrooms," in *Video Proceedings of ACM SIGCHI'97,* ACM Press, Atlanta, Georgia, March 22-27 1997.

[26] Swenson, K.D., Maxwell, R.J., Matsumoto, T., Saghari, B., and Irwin, K., "A Business Process Environment Supporting Collaborative Planning," *Journal of Collaborative Computing*, vol. 1, no. 1.

[27] TeamWARE Inc, *TeamWARE Flow*, http://www.teamware.us.com/products/flow/, 1996.