Preprint of paper published in Journal of End User Computing, © Idea Group Publishing 1998.

SUPPORTING LARGE–SCALE END USER SPECIFICATION OF WORKFLOWS, WORK COORDINATION AND TOOL INTEGRATION

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

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

††Department of Computer Science University of Auckland
Private Bag, Auckland, New Zealand

Abstract

Workflow Management Systems are a common example of an end user development system, in that they support end user specification of work process models, end user enactment (running) of these models, and end user evolution of workflows. Example applications of Workflow Management Systems include office automation, software process modelling and business process codification. We describe a novel workflow system which provides visual support for end users to specify both simple and complex aspects of workflows, including workflow models, communication and collaboration techniques, and the tools used to perform work. We illustrate both the utility of these work coordination mechanisms and the degree of scalability they provide for both novice and experienced end users. We also outline the architecture of our system, our experiences with this workflow tool, and future research directions in this area.

1. Introduction

Many software systems now support varying degrees of end user development. Many systems, such as MS Word™, provide users with complex facilities to dynamically configure their tools; Web browsers, such as Netscape™, provide users with mechanisms for integrating third-party software applications; and cooperative work systems, such Lotus Notes™ and TeamFLOW™, allow multiple users to coordinate their work with others. Due to the complexity of the work environments these and most other software systems are deployed, end users often desire (or even require) mechanisms for dynamically specifying their work processes, coordination mechanisms with other people, tools and processes, and the set of tools and tool interactions they require in order to most effectively perform their work.

Workflow Management Systems (WFMSs) and Process-Centred Environments (PCEs) have been developed to enable end users to specify just such aspects of their work and work environments. They have become popular for use in various fields, including office automation, business process reengineering and software development (Bandinelli et al, 1996; Medina-Mora et al, 1992; Swenson et al, 1994). Such systems typically allow users to model work processes using graphical and textual notations, and then run ("enact") these process models to guide (or enforce) work on particular projects. Most workflow and process modelling notations and tools permit users to specify steps in their work processes, the artefacts and tools used to perform each step, and the other users and tools they need to interact (coordinate) with during work.Workflow models are often reused on different projects and refined (improved) over time.

Workflow systems are typically used in conjunction with other tools for performing work, such as document editors, information systems and software development tools (Bandinelli et al, 1996; Medina-Mora et al, 1992). Often multiple people collaborative on a project and thus use these tools and the workflows (Swenson et al, 1994). This requires coordinating the work done with both the tools for performing work and the workflow tools themselves. Often the work coordination approaches, the workflow models and the work tools used on a project, evolve over time. Thus end users of workflows

require support for not only modelling, enacting and evolving workflows, but support for coordinating their work with others and integrating a variety of different tools into their work processes.

Unfortunately most existing workflow and process modelling systems do not provide sufficient support for specification of workflows, work coordination mechanisms and tool usage by end users, or provide overly-complex facilities which many end users can not use. Many systems utilise complex, textual languages to specify even simple coordination and tool integration mechanisms, which are difficult for many kinds of end users to understand or modify (Swenson et al, 1994; Grundy and Hosking, 1998). Many workflow notations and tools do not adequately support work coordination and tool integration, relying on limited form-based facilities for specifying only some of these aspects of cooperative work.

We describe a workflow/process modelling environment we have developed which supports the specification of simple and complex workflow models, work coordination schemes and tool usage, by both novice and expert end users. This uses a graphical notation to describe both workflow (work process) models and the handling of events to support work coordination and tool integration. This paper focuses on the utility of our approach for end user specification of various work coordination activities. We also compare its support to other workflow/process modelling systems, and briefly describe its architecture, our experiences using the tool, and our future research directions.

2. Serendipity

Serendipity is a process modelling, enactment and work planning environment, which also supports flexible event handling, group communication, and group awareness facilities (Grundy and Hosking, 1998; Grundy et al, 1996a). Serendipity's notations are designed to be high-level and graphical in nature, and its coordination and rule mechanisms be easily extended by end users. Key aims during our development of Serendipity were: i) to produce notations that are suitable for both simple workflow and work coordination tasks, and yet scale up to large, complex process modelling tasks, and ii) notations which are simple enough for novice end users to understand and utilise, but powerful and expressive enough for expert end users to extend the environment and workflow model capabilities as required.

The left and bottom windows shown in Figure 1 are Serendipity views modelling part of the ISPW6 software process example (Grundy et al, 1996a). Stages (rounded rectangle icons) describe steps in the process of modifying an arbitrary software system, with each stage containing a unique id, the role which will carry out the stage, and the name of the stage. Enactment event flows link stages, labelled with the finishing state of the stage the flow is from. There are a number of specialised types of stage including start, finish, AND, and OR stages. Modularity is provided in the form of hierarchical subprocess models. The window at the left of Figure 1 is a subprocess model refining the "ispw6.2:Design, Code & Test" stage of the process in the bottom window. Underlined stage IDs/roles mark the presence of a subprocess model. Serendipity supports artefact, tool and role modelling for processes. Usage connections show how stages, artefacts, tools and roles are used. Optional annotations indicate: data is created (C), accessed (A), updated (U), or deleted (D); whether a stage must use only tools, artefacts or roles defined (√); and whether a stage cannot use specified tools, artefacts or roles (¬).

In addition to specifying the static usages and enactment event flows between process model stages, Serendipity supports filters (rectangular icons) and actions (ovals), which process arbitrary enactment and work artefact modification events. For example, the coordination of the process model via the "ispw6.3:Monitor Progress" stage is defined by the top-right window of Figure 1. This uses two filters and three actions to carry out the coordination. The Enacted filter selects only stage enactment events, in this case when the ispw6.2 stage is enacted. This triggers the "Notify Changes Started action", which notifies its associated role (in this case, the project manager) that changes have commenced. The other filter acts similarly to notify commencement of testing. The other action takes artefact modification events from the OOA/D design document and accumulates them into a changes summary. Serendipity models may be used to guide work or to enforce particular work processes (by defining rules with filters and actions). Complex filters and actions can be implemented by either reusing other, simple filter/action models, or by using a textual API interface from Serendipity to its underlying implementation language (Grundy and Hosking, 1998).

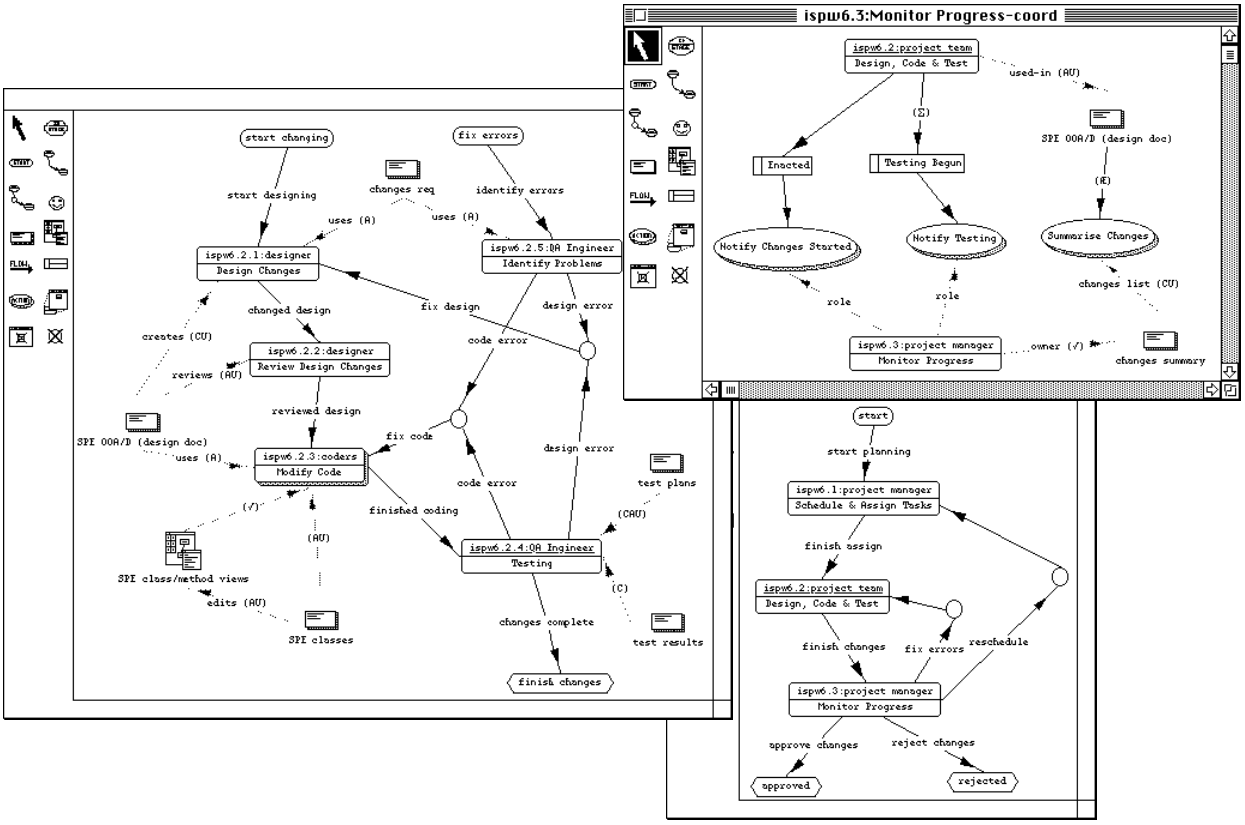

Figure 1. Part of the ISPW6 software process example modelled in Serendipity.

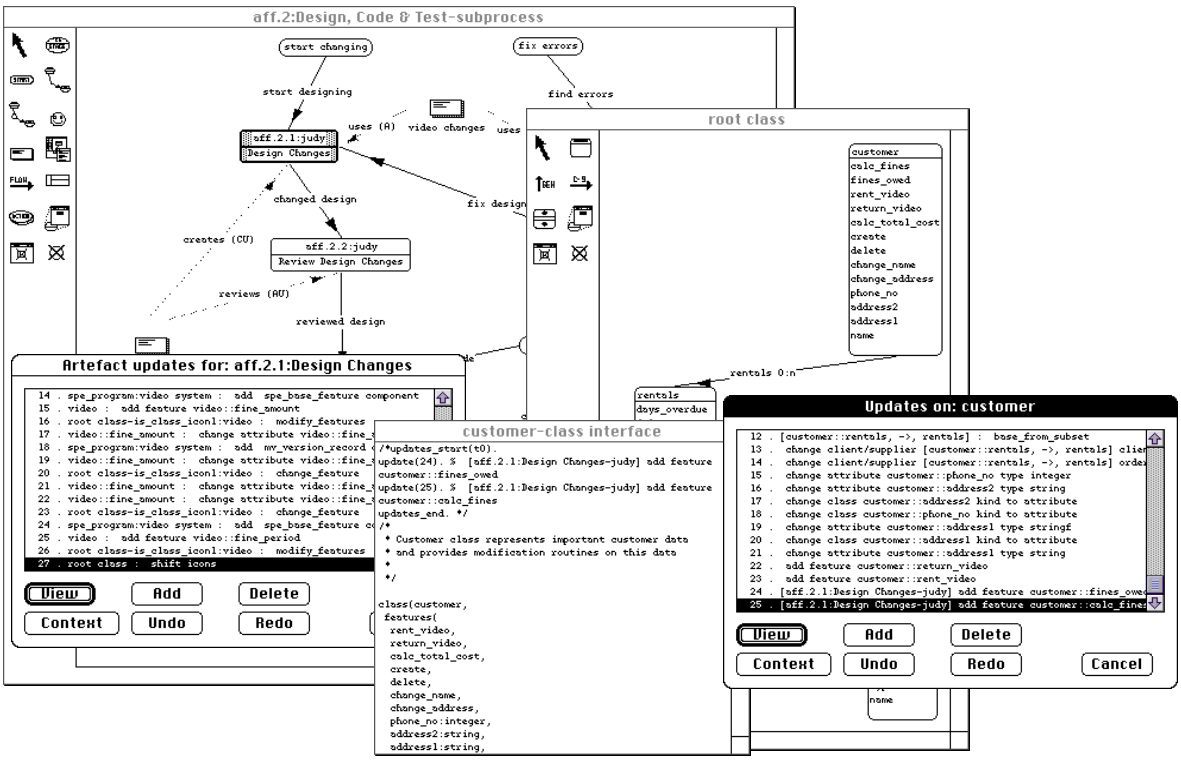

Figure 2. An example of work context capture and presentation in SPE–Serendipity.

Serendipity has been integrated with software and information system development tools (Grundy and Hosking, 1998; Grundy et al, 1996a; Grundy and Venable, 1996), office automation tools (Grundy and Hosking, 1998), and a variety of CSCW tools (Grundy and Hosking, 1998; Grundy et al, 1996a). This was done by using the filter/action language to specify event propagation mechanisms between Serendipity artefacts (stages etc.) and the repositories of these other tools. Serendipity tool and artefact icons can be linked by end users to appropriate other tool artefacts and repositories, producing interfaces from Serendipity workflow and filter/action models to these external systems.

Events propagated to Serendipity from integrated tools are utilised in various ways to support work history determination for process stages, work coordination between multiple users and tools, and to realise various tool integration mechanisms. Figure 2 shows Serendipity being used with an integrated software development environment, called SPE (Grundy et al, 1995a, Grundy et al, 1996a). The Serendipity ISPW6 process model from Figure 1 has been tailored for use on this particular project and guides multiple developers' work on the software. Enacted parts of process models are shown by shaded icons and event flows. Changes made in the software development tools stored against process stages, and process stage information augmenting artefact update descriptions in the software tools.

The following sections focus on Serendipity's support for modelling, enacting and evolving workflow models, its support for both simple and complex kinds of end user-specified work coordination, and its support for describing tool usage and integration with workflows. We focus on both the support provided by Serendipity for these purposes, how different kinds of end users can make use of this support, and scalability issues of our notations and their support environment.

3. Workflow Development

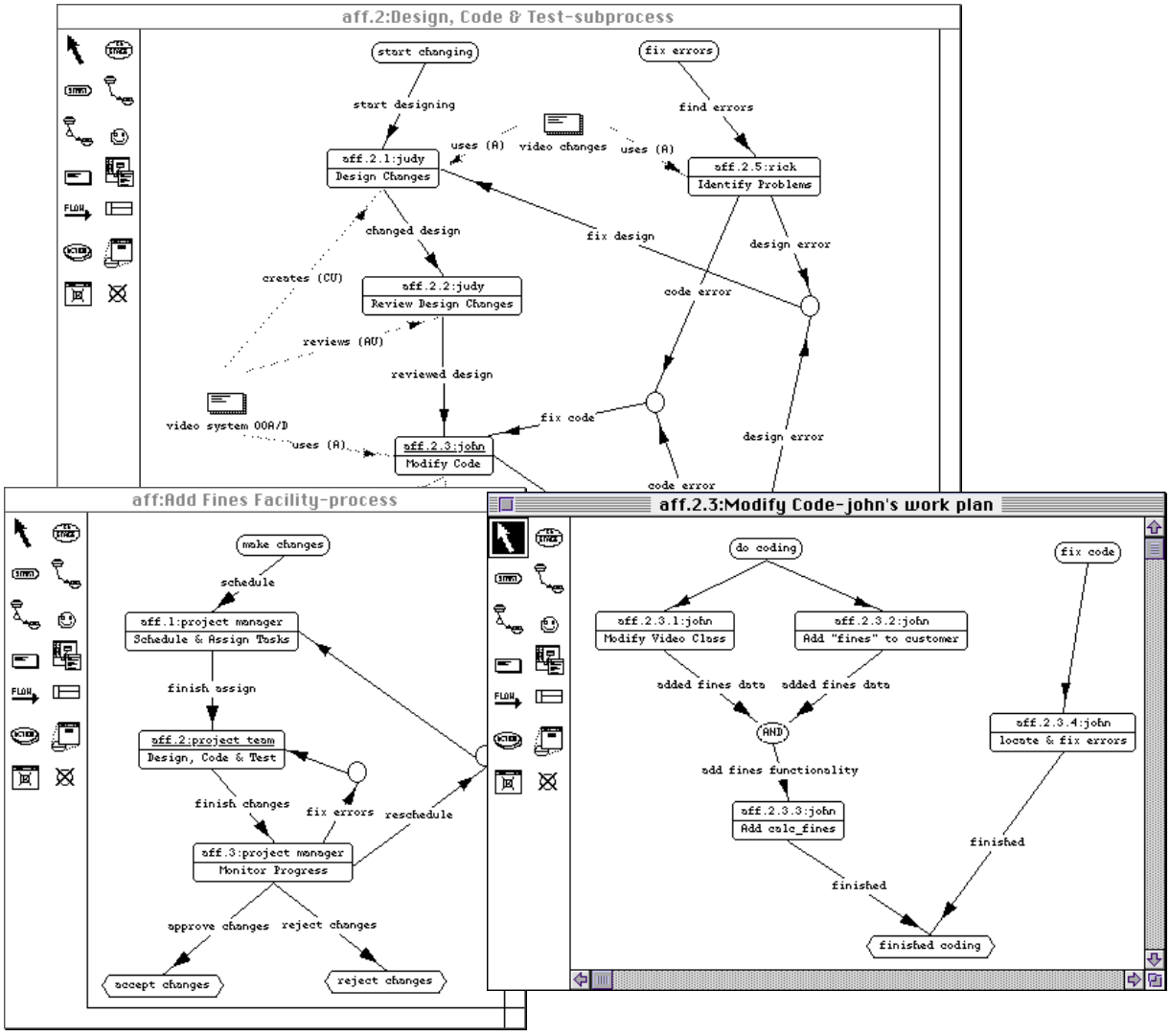

Figure 3. A reused template and expanded work plan for a particular project.

End users can readily model, enact, reuse and improve Serendipity process models, due to their primarily graphical nature. For example, the process models in Figure 3 were reused from the ISPW6 software process model template shown in Figure 1. This reused model is to guide the work to be done for modification of a video library program in SPE. The idea is to add a fines facility to the software, and have Serendipity guide the work done in SPE and record all changes made to the software.

The project manager has copied the ISPW6 template and changed the process stage prefixes to “aff” ("Add Fines Facility"). The abstract names for developers have been changed to those who are filling these roles for this particular project using this workflow template. Search and replace facilities are provided by Serendipity to ease this task. A project-specific “work plan” for coder “john” has also been developed by the project manager (bottom right window). Other end users can also develop and modify their own plans, and the group can collaboratively modify any part of the workflow model as a whole. Changes to this project-specific workflow model can be abstracted back into the reusable template to improve it. Figure 4 shows an example of changes made to the add2.1 subprocess being merged back into the iswp6.2.1 subprocess to improve this reusable template.

Serendipity supports both synchronous, “what you see is what I see” editing of workflow diagrams by multiple users, as well as merging changes made to asynchronously (indendently) modified models. Such multi-user viewing and editing of workflows is required in order for Serendipity workflows to be used in complex, multi-user work environments.

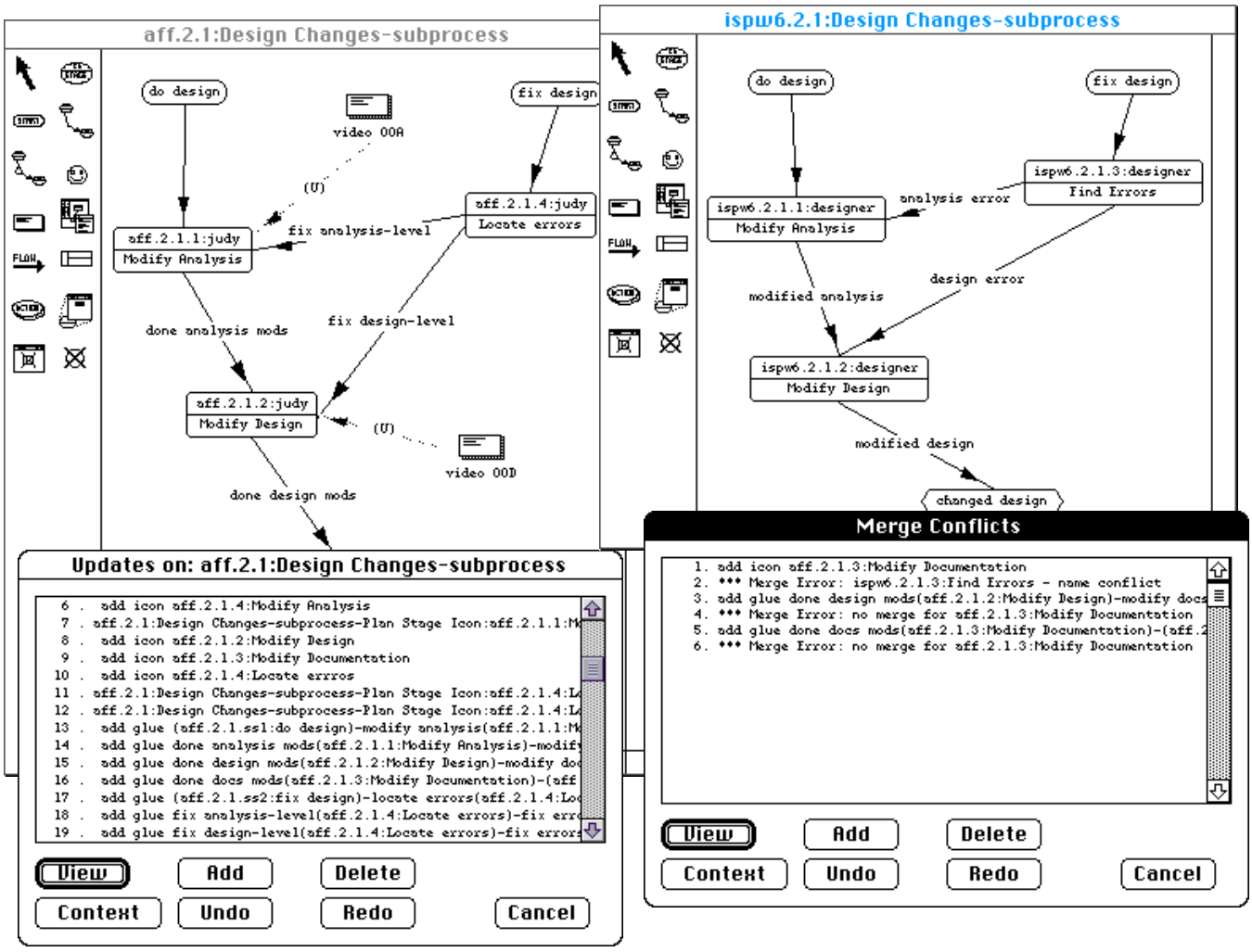

Figure 4. Independant modification and merging of workflow models.

The Serendipity workflow notation and support environment allow both novice and experienced end users to understand, reuse and modify both simple and complex workflow systems. Detailed information can be specified about a model using textual forms and views, but most information about the work processes being modelled are captured in the graphical diagrams. The visual notation and support environment allow novice users to experimentally build and enact (even incomplete) workflows, adding more information and complexity as they refine their work processes. Experienced users can specify complex process stage interactions, tool and artefact usage, and inter-person communication. The notation’s simple conceptual model of “enactment flow” between stages dividing up steps in a work process, is both simple enough for novice users to understand and use, but allows very complex, interacting workflow models to be developed by experts.

The notations and support environment scale up from modelling simple workflow problems to use on large, complex problems. Serendipity not only provides multiple view support with consistency management for workflow models, but multi-user editing and merging capabilities, search facilities, libraries of reusable templates, and dialogues for viewing enacted process stages, stage statistics, enactment histories etc.

5. Work Coordination

Figure 1 showed an abstract work coordination view built using Serendipity filter/actions to keep the project manager informed of progress on a project using this workflow model. Filter/actions were used to detect changes made to artefacts when particular process model stages were enacted, and the project manager is informed of these in various ways. Other work coordination schemes can be developed using our filter/action language to support simple or complex work coordination using workflow models and integrated tools.

For example, in Figure 5 coder "john" has defined a new event-handling view in Serendipity and added filter/actions to keep him aware of modifications done by "judy" on the artefact "video class" (the artefact changes are stored in a "change list" artefact), and to inform him when "rick" starts work on "test_unit". In this example, changes judy makes are stored and thus john would browse this modification history at a later date, or have a dialogue showing these changes automatically updated for him. In contrast, when rick starts testing, john is informed immediately by a broadcast message.

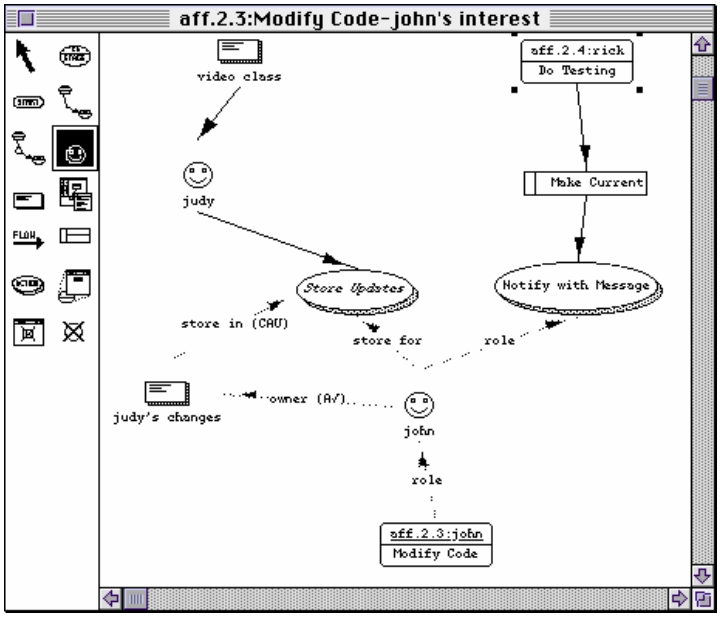

Figure 5. Extra actions specified to keep people aware of other's work.

The conceptual basis of Serendipity's event handling language is one of filtering "interesting events" (including stage enactment, artefact update, tool and communication events), and then carrying out actions in response to the filtered events. This visual language has proved to be useful and accessible for novice users of Serendipity, allowing them to build simple work coordination schemes. The work coordination examples in Figures 1 and 5 are simple enough for novices to both understand and to build for themselves for use in similar situations. Our filter/action language has also proved to be powerful enough for experienced users to build reusable, complex event handling models to implement a range of diverse work coordination facilities (Grundy et al, 1997b; Grundy and Hosking, 1998).

Figure 6 shows an example of a filter/action model, built by an expert end user, being used to highlight icons in Serendipity views, to indicate access/update of artefacts and use of tools. Other users' enacted stages are highlighted, in addition to the user who is being shown this view (judy). Artefacts and tools in use by judy and her collaborators are highlighted. This was achieved by storing recent artefact modification and tool events, and filter/actions use this history to highlight appropriate icons to support group awareness (Grundy et al, 1997b).

While the definition of such filter/actions is complex for more inexperienced end users to build, such novice end users can easily reuse the packaged filter/action behaviour on their own Serendipity process model views. Users select the highlighting filter/actions they require, e.g. "example2–serendipity_highlight" as in Figure 6, and attach these to artefacts representing Serendipity views and tools (i.e. to meta-level representations of Serendipity artefacts). End users can modify template visualisation filter/actions to suit their own awareness needs, due to the accessibility and power of expression of our visual event handling language. Highlighting and constraint filter/actions can also be applied to other tools integrated with Serendipity, as described in the following section.

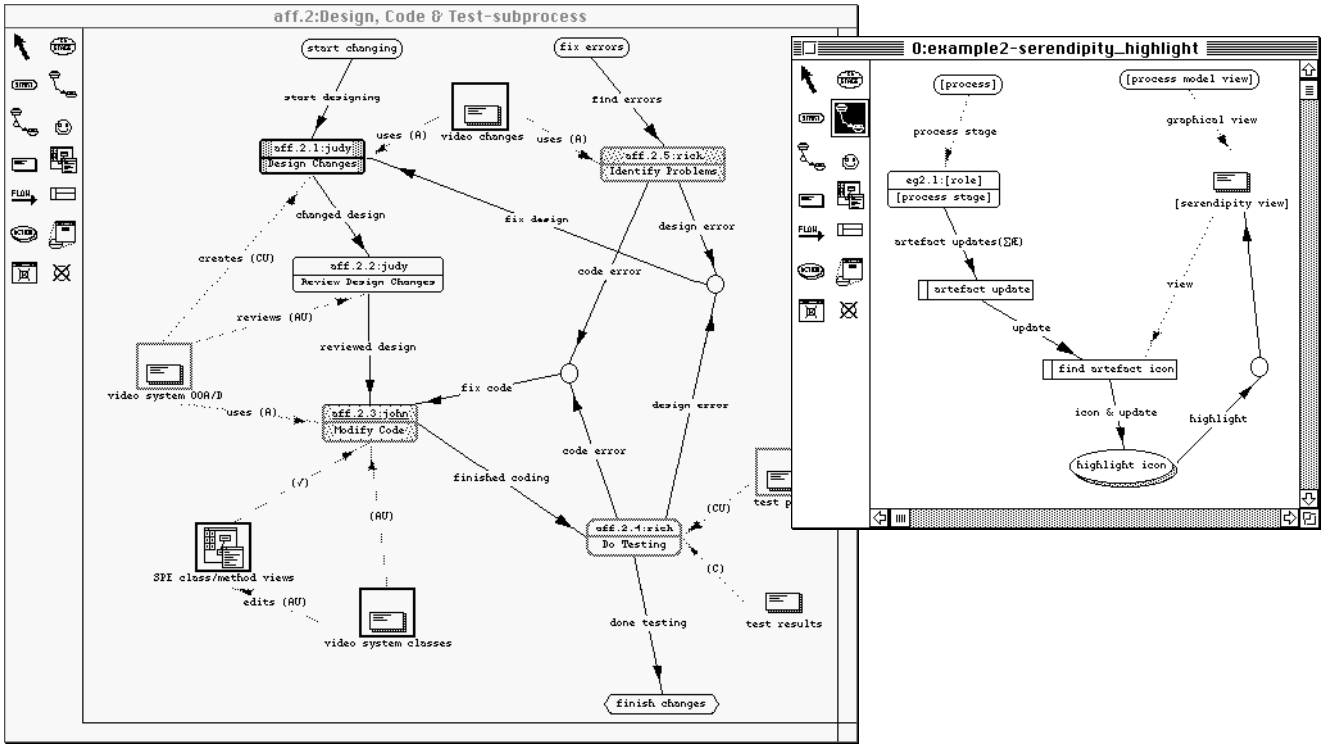

Figure 6. Highlighting icons to support awareness.

The specification of work coordination schemes is one of the more poorly-supported aspects of most existing WFMSs and PCEs. Most utilise form-based interfaces or textual languages to allow users to specify simplistic event handlers, which provide a basic mechanism to inform users of interesting event occurances or coordinate multiple process usage. In contrast, Serendipity provides a visual language which can be used by novice users to build simple event handling models, using a wide range of template filter/actions. In addition, expert end users can build highly sophisticated filter/actions, even using an API to the implementation language of Serendipity, to build very complex work coordination schemes. These filter/actions can then be easily reused by the experts, or novices, by their packaging into single, parameterised filter/action icons in the Serendipity visual event-handling language.

5. Supporting Tool Integration

Tools ranging from those built with the same underlying architecture as Serendipity to third-party tools can be integrated with our workflow environment, with differing levels of integration completeness. SPE, other software and information system development tools, and a variety of small Computer-Supported Cooperative Work (CSCW) tools have been tightly integrated with Serendipity (Grundy and Venable, 1996; Grundy et al, 1996a). Figure 7 shows how integrated messaging and annotation tools, as well as SPE, can be integrated with Serendipity and utilise its process model information. Descriptions of changes in SPE are augmented with enacted process stage information to help better describe why and when changes were made in SPE (examples of this are also shown in Figure 2). The messaging and annotation tools are sent Serendipity process model information which is incorporated into the sent messages and stored notes respectively.

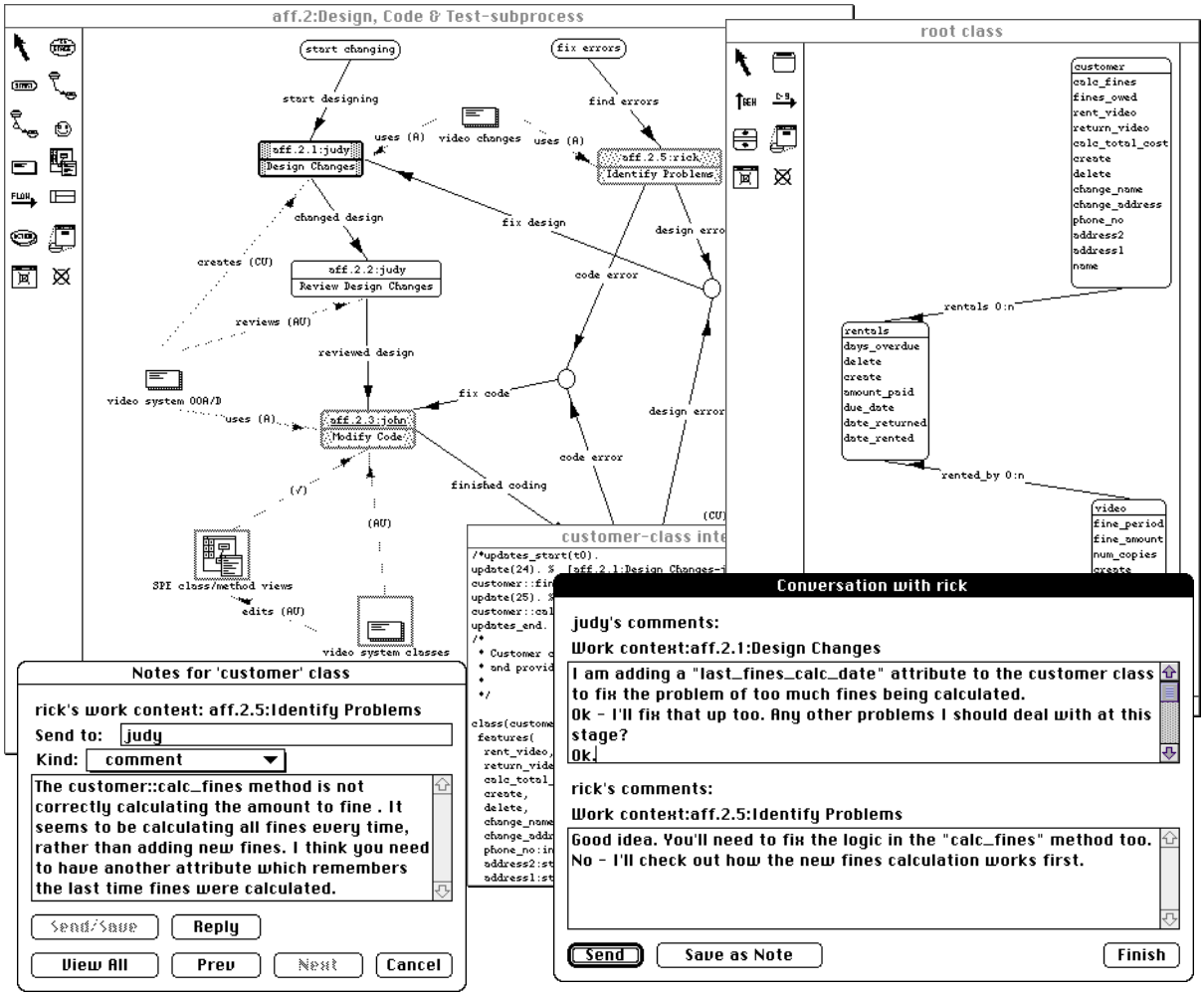

Figure 7. Examples of work context awareness and communication support.

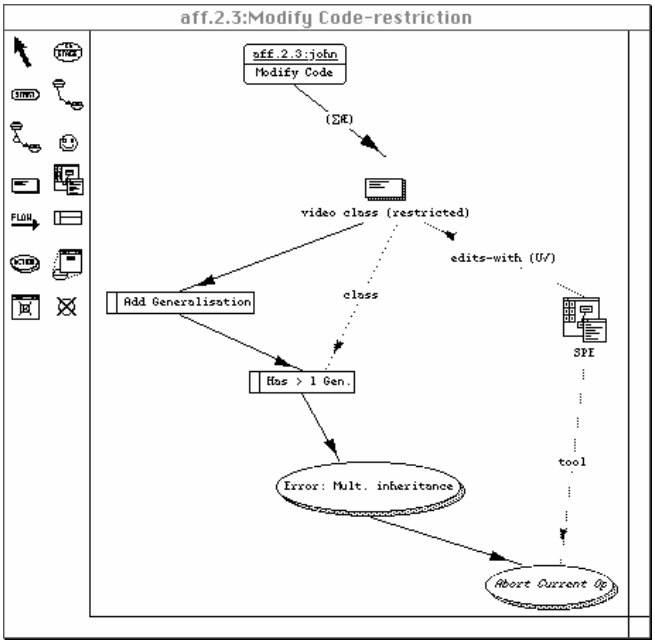

Figure 8. Constraining integrated tools with Serendipity filter/action models.

With tightly integrated tools like SPE, Serendipity filter/actions can even be used to constrain the behaviour of the tools. Figure 8 illustrates how SPE tool actions may be restricted by a software process model, illustrating a simple form of Method Engineering (Harmsen and Brinkkemper, 1995). The video class is prevented from using multiple inheritance, thus extending the semantics of SPE itself by using Serendipity filters and actions to process SPE artefact modification events. We have also built

considerably more complex filter/actions and workflow models to support more comprehensive Method Engineering for a range of Information Systems Engineering tools (Grundy and Venable, 1996).

Figure 9 shows Serendipity being used to support process modelling for a suite of office automation programs, including Macintosh versions of Microsoft Word™, Microsoft Excel™, the GlobalFax™ fax/OCR application, and the Eudora™ email utility. Two simple Serendipity process models describing semi-automated processes of ordering and receiving stock for an organisation. Actions are used to launch the programs and to create, open, and save files via Apple Events sent from Serendipity to the running programs.

Third-party tools, like those shown in Figure 9 can not be constrained or integrated to the same degree as our tools built with an open message-passing architecture (Grundy et al, 1996b, Grundy et al, 1996c). However, we have built up useful office automation system workflows and work coordination mechanisms, as illustrated in the Figure 9 example. Simple communication with these tools is supported by a range of template filter/actions to invoke tools, tell tools to open/close/save files, and to tell tools to exit. We have also developed filters which detect when tools exit or when they open, close and update files, generating events that end user defined filter/action models can make use of.

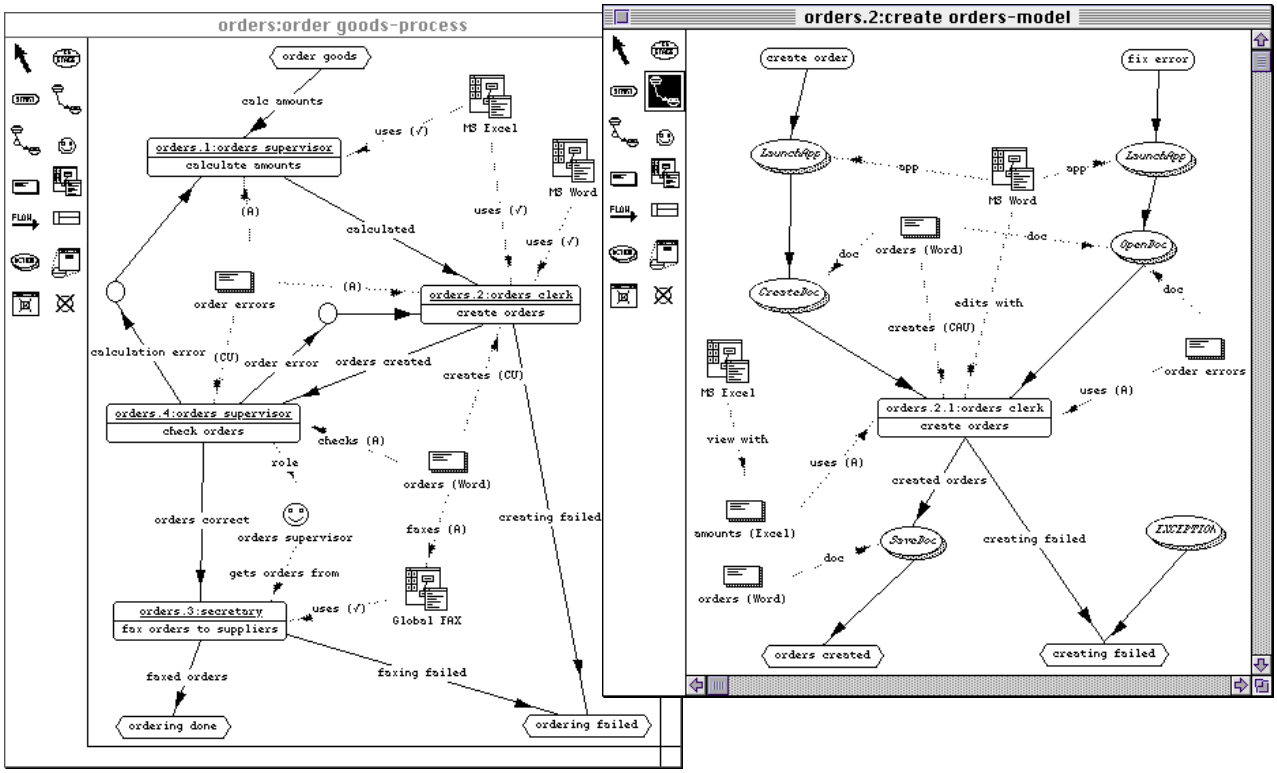

Figure 9. A simple office automation process described in Serendipity.

6. Architecture

Serendipity is currently implemented using the MViews Prolog-based architecture for building multi-view, multi-user editing tools (Grundy and Hosking, 1996). MViews provides a general model for defining software system data structures and tool views, with a flexible mechanism for propagating changes between software components, views and tools. ISDE data is described by *components* with *attributes*, linked by a variety of *relationships*. When a component is updated, a *change description* is generated. Change descriptions are propagated to all components 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 propagation mechanism supports a diverse range of software development environment facilities, including attribute recalculation, multiple views with flexible, bi-directional textual and graphical view consistency, a generic undo/redo mechanism, component versioning, and collaborative view editing (Grundy et al, 1996b).

Environments such as SPE are integrated with Serendipity by instructing SPE's repository to forward all events generated by SPE tools or artefact modifications onto Serendipity, which then forwards the event to the currently enacted process model stage. This mechanism is used to build work histories within Serendipity for each process model stage, and to annotate SPE and other environment events with process model stage information (i.e. why a change was made or event generated). A similar approach has been used with CSCW tools, such as note annotations and textual chats, to indicate the process stage a note or chat is focused on.

Rather than alter the implementations of Serendipity and SPE directly to provide the awareness visualisation techniques described in the previous Sections, we have used Serendipity's filter/action language to implement them. This has the advantage of not having to modify the source code of the environments, allows end users to look at how the various visualisation etc. utilities have been implemented, and because of the high-level nature of our filter/action language, this allows end users to tailor these to their own needs.

A simple "work history" determining filter/action is shown in Figure 10. This is used to collect artefact, tool and communication events generated by the user's actions, are store these in a work history artefact (right-hand event flow). Any "set current stage" enactment events cause this work history to be partitioned, grouping artefact etc. events according to which stage the developer currently had enacted. Process model users can modify this filter/action to only record particular kinds of events, or to record them in different work history artefacts (Grundy et al, 1997b).

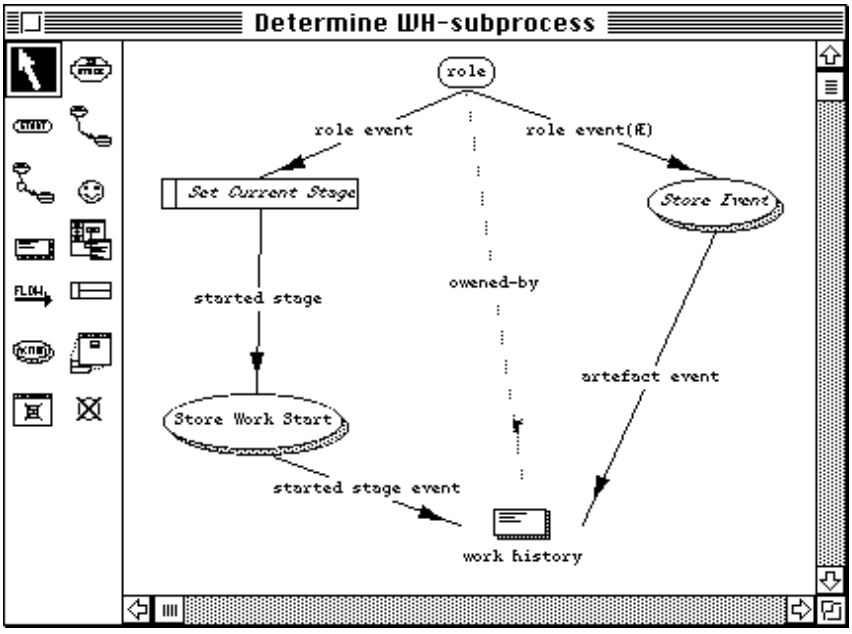

Figure 10. A Simple filter/action determining work histories.

7. End User Experience

We have had considerable success with using our workflow and event-handling notations for small and medium-sized software process, method engineering and office information system applications. Both novice and expert end users can utilise the notations to configure their work coordination schemes and to make use of other tools from within their workflow models. Serendipity provides additional support tools for searching for specified keywords, recording events to manage inconsistencies between views and enactment events for stages, abstracting workflow and filter/action models into reusable templates, and for grouping templates for browsing and reuse.

The power of expression of Serendipity's workflow and event handling languages for novice users, and the accessibility of the notations for these users, has proved reasonable. There is some confusion when novices try and specify filter/actions using multiple event sequeneces, and when they use usage link

annotations in both languages. Expert users can build substantial workflow models using the workflow notation, and quite complex work coordination, tool usage specifications and tool integration schemes with the event handling notation. To use facilities of the Serendipity implementation which aren't directly supported by existing filters or actions they must use the textual API language (Prolog). However, once complex filter/actions have been built using the hierarchical graphical language or textual API, they can be packaged for reuse by both novices and experts as single, reusable icons.

The multi-user support within the current version of Serendipity, and its implementation platform, MViews, is not sufficient for complex, multi-user workflow-based systems. The performance of the multi-user aspects of Serendipity need to be made both more responsive and more robust to allow the environment to be deployed within "real-world" complex work environments. In addition, the tool integration approaches for third party tools (i.e. those not built with MViews) are currently inadequate, as few tool events and interactions can be handled by the current Serendipity filter/actions which interact with these third party tools.

We are continuing to improve our notations, adding improved modelling capabilities for expressing function calling between filter/action icons and for expressing specialised forms of event filtering, tool repository querying and better supporting tool integration. We are also developing notational extensions for better expressing multiple event filter/actions, where event sequencing can be expressed using a visual finate state machine representation. We are also making compilation improvements to the efficiency of the filter/action models, which are currently interpreted. These improvements are required to help Serendipity be more extensively used in larger workflow-based systems.

8. Future Work

Due to the problems mentioned above that we encountered when trying to deploy Serendipity on large workflow systems, we are currently porting Serendipity to JViews, a Java-based successor to MViews (Grundy et al, 1997a). This will allow a more open-architecture and portable workflow environment to be developed, and mean third party tools can much more effectively be integrated with Serendipity. The filter/action language from Serendipity has been reimplemented as a stand-alone tool for processing events from JViews components, and is part of the JComposer componentware environment for building JViews-based systems (Grundy et al, 1997a). Parts of these filter/action models can be compiled and thus execute much more quickly than the Serendipity versions, making them more powerful for a wider range of end user programming tasks.

Figure 11 shows a running Entity-Relationship modeller generated using JComposer, and our new event-handling language being used to provide work coordination support. The visualised JViews components (top left window) can be used in an event-handling view (top right window) to specify the presentation of events of interest to a user (in this case the user of this tool). The bottom-right window shows our visual query language being used to query the tool repository (in this case to find all entities with a specified name which have links to at least one relationship).

We are continuing to develop JComposer and its support tools, and will utilise the JComposer event handling language and visual query language in a port of Serendipity. We are also building components to provide better integration support for third-party tools, based on the Java Beans componentware API. This will allow end users more control over a wide variety of tools used with JComposer and Serendipity, and thus support better end user specification of work coordination and tool integration mechanisms.

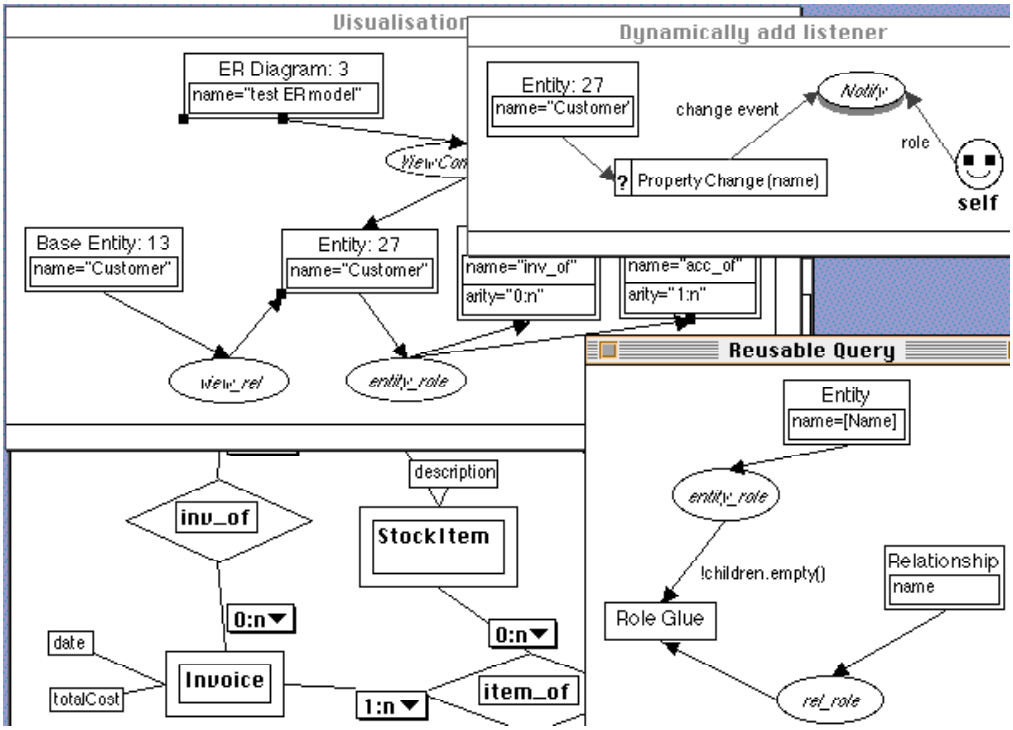

Figure 11. Visualising running tool data and extending tool event handling with JComposer.

9. Related Work

Much research into Computer Supported Cooperative Work (CSCW) systems has focused on low-level interaction mechanisms, such as synchronous and asynchronous editing. Examples include most Groupware systems (Ellis et al, 1991), GroupKit (Roseman and Greenberg, 1996), Mjølner (Magnusson et al, 1993), C-MViews (Grundy et al, 1995b), and Rendezvous (Hill et al, 1994). These systems generally lack support for end user tailoring of work coordination mechanisms, as provided by Serendipity. Our experiences with such systems suggest that the techniques provided by Serendipity for coordinating work, allowing end users to specify work coordination and group awareness mechanisms, and integrating third party tools are essential to make CSCW tools organisationally feasible.

Some work has been done on providing higher-level process modelling and coordination facilities, such as workflow configuration in Action Workflow (Medina-Mora et al, 1992), TeamWARE Flow (Teamware Inc., 1996), and VPL (Swenson et al, 1994), software process protocols in ConversationBuilder (Kaplan etl al, 1992) and Oz (Benshaul and Kaiser, 1994), and various kinds of shared workspace awareness in GroupKit (Roseman and Greenberg, 1996). Once again, these approaches generally do not provide the level of end user configuration of work coordination and awareness mechanisms, as provided by Serendipity, which are required in large workflow-based systems. Most workflow management systems, such as Action Workflow (Medina-Mora et al, 1992), Regatta (Swenson et al, 1994) and TeamFLOW (Teamware Inc., 1996), and groupware tools like Lotus Notes™, provide a limited range of "interesting events", supporting interaction with other tools and some forms of notification and work coordination based on event occurrence. These are usually codified using a form-based approach where modellers specify simple actions to carry out based on a range of possible events. Our visual event handling language provides a much wider range of capabilities, but remains accessible to both novice and expert end users. It also allows expert users to build very complex filter/action models supporting work coordination and tool integration, but package these for novice end users to easily reuse.

Many Process-Centred Environments (PCEs), such as SPADE (Bandinelli et al, 1996), Merlin (Peuschel et al, 1992), Marvel (Barghouti, 1992), Oz (Ben-Shaul and Kaiser, 1994), CPCE (Lonchamp, 1995), and EPOS (Jaccheri et al, 1992), use complex, textual descriptions of processes and tool configurations which are often very difficult for end users to understand and modify, often can not be modified while in use, and have poor exception handling. The event-handling of most graphical process modelling languages is

codified graphically for "enactment" events (state transitions), but textually for stage guards and actions, and for coding interaction with people or other tools. ProcessWEAVER (Fernström, 1993) provides a textual co-shell language which allows users to specify actions for process model nodes when fired by input tokens. SLANG (Bendinelli et al, 1996) uses textual specifications of guards and actions for nodes in a state transition network. Marvel and Oz (Ben-Shaul and Kaiser, 1994) use guarded rules specified over data types. Adele (Belkhatir et al, 1994) provides an activity manager, which uses a textual language to specify database-related event handling for process models. These approaches all use textual languages, and hence all but expert end users find such specifications difficult to understand, modify and reuse. In contrast, our visual language-based approaches are much more accessible to both novice and expert end users, but are expressive and powerful enough for experts to greatly extend the environment's capabilities.

Visual dataflow-based languages, such as Fabrik (Ingalls et al, 1998) and Prograph (Cox et al, 1989), provide graphical dataflow models which are similar in nature to our event-handling language, but use dataflow rather than event-flow, which tends to be less appropriate and useful in a workflow and tool integration domain (Swenson et al, 1994). Some visual languages, such as ViTABaL (Grundy and Hosking, 1995), utilise an event-driven model but lack the equivalent of Serendipity's filters, actions, and interest specification capabilities. Because of their general-purpose nature, these visual programming languages lack specific workflow modelling capabilities, and thus can not express and represent process model event-handling and work coordination tasks as effectively as Serendipity's languages.

Many workflow tools and PCEs are generally not well integrated with existing tools used to perform work Bendinelli et al, 1996, Krishnamurthy and Hill, 1994, Marlin et al, 1993). They thus can not achieve the same degree of utilisation of high-level workflow and process model information to augment work tool and low-level CSCW capabilities as supported by Serendipity. Some work, including SPADE/ImagineDesk (Bandinelli et al, 1996), ConversationBuilder (Kaplan et al, 1992), MultiviewMerlin (Marlin et al, 1993), wOrlds (Bogia and Kaplan, 1995), and Oz (Valetto and Kaiser, 1995), attempts to bridge the gap between workflow/PCEs and CSCW. So far these have had some success, but there are continuing problems of integrating existing tools into the environments, effectively utilising workflow/process model information in the integrated tools, and with limitations of the process modelling tools used (Bogia and Kaplan, 1995; Marlin et al, 1993; Valetto and Kaiser, 1995). Our integration of MViews-based tools with Serendipity has been very successful, and our filter/action language can even allow such tools to be constrained (Grundy et al, 1996a). With more open-architectured third party tools becoming available with our JViews-based version of Serendipity, improved integration of these tools should also be possible.

10. Summary

We have described visual languages and a support environment for the end user specification of workflows, work coordination mechanisms and tool integration in workflow systems. Our notations allow both novice and expert end users to readily model their work processes, to specify both simple and complex event-handling for these models, and to integrate a variety of tools for both performing work and communicating with other users into their workflow models. We have had success using these end user programming techniques in a variety of application domains, including software process modelling, software development, method engineering, CSCW applications and office automation systems. Our notations, while being accessible to novice end users, are also powerful enough for expert end users to build very sophisticated work coordination and tool integration facilities. The languages are also useful in both small-scale, single-user workflow systems, and larger, multi-users work environments.

We have ported a modified form of our event-handling language to Java, and are continuing to redevelop our workflow, CSCW and software development tools. This Java Beans-based implementation will allow our techniques for work coordination specification and tool integration to be more effectively used with wider range of third-party tools in complex application domains. We are also continuing to develop both the workflow and event-handling notations of our environments, to preserve and extend the accessibility to all kinds of end users, while improving their power and usefulness in diverse application domains.

References

Bandinelli, S., DiNitto, E., & Fuggetta, A. (1996). Supporting cooperation in the SPADE-1 environment, *IEEE Transactions on Software Engineering*, 22 (12), December, 1996.

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

Belkhatir, N., Estublier, J., and Melo, W.L. (1994). *The Adele/Tempo Experience*, Software Process Modelling & Technology. Finlelstein, A., Kramer, J. and Nuseibeh, B. Eds, Research Studies Press.

Ben-Shaul, I.Z. and Kaiser, G.E. (1994). A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment, in *Sixteenth International Conference on Software Engineering,* May, 1994, pp. 179-188.

Bogia, D.P. and Kaplan, S.M. (1995). 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.

Cox, P.T., Giles, F.R., and Pietrzykowski, T. (1989). Prograph: a step towards liberating programming from textual conditioning, in *Proceedings of the, 1989 IEEE Workshop on Visual Languages,* IEEE Computer Society Press, pp. 150-156.

Ellis, C.A., Gibbs, S.J., and Rein, G.L. (1991). Groupware: Some Issues and Experiences, *Communications of the ACM*, vol. 34, no. 1, 38-58, January, 1991.

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

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

Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. (1995b). 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.

Grundy, J.C. and Hosking, J.G. (1995). ViTABaL: A Visual Language Supporting Design By Tool Abstraction, in *Proceedings of the, 1995 IEEE Symposium on Visual Languages,* IEEE CS Press, Darmsdart, Germany, September, 1995, pp. 53-60.

Grundy, J.C., Hosking, J.G., and Mugridge, W.B. (1996a). Low-level and high-level CSCW in the Serendipity process modelling environment, in *Proceedings of OZCHI'96,* IEEE CS Press, Hamilton, New Zealand, Nov 24-27, 1996.

Grundy, J.C. and Venable, J.R. (1996). Towards an environment supporting integrated Method Engineering, Proceedings of the IFIP 8.1/8.2 Working Conference on Method Engineerig, Atlanta, August 26-28, Chapman-Hall, 1996.

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

Grundy, J.C., Hosking, J.G., Mugridge, W.B., and Amor, R.W., (1996c). Support for Constructing Environments with Multiple Views, in *Joint Proceedings of the SIGSOFT'96 Workshops,* ACM Press, San Francisco, October 14-15, 1996, pp. 212-216.

Grundy, J.C. and Hosking, J.G. (1996). Constructing Integrated Software Development Environments with MViews, *International Journal of Applied Software Technology*, Vol 2, No. 3/4, 1996.

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

Grundy, J.C., Mugridge, W.B., and Hosking, J.G. (1997b). Utilising past event histories in a process-centred software development environment, in *Proceedings of the, 1997 Australian Software Engineering Conference,* Sydney, Sept. 30-Oct 2, 1997.

Grundy, J.C. and Hosking, J.G. (1998). Serendipity: integrated environment support for process modelling, enactment and improvement, Automated Software Engineering: Special Issue on Process Technology, vol. 5, no. 1, January, 1998, Kluwer Academic Publishers (in press).

Harmsen, F., and Brinkkemper, S. (1995). 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, pp. 430-438.

Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F., and Wilner, W. (1994). The Rendezvous Architecture and Language for Constructing Multi-User Applications, *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, 81-125, June, 1994.

Ingalls, D., Wallace, S., Chow, Y.Y., Ludolph, F., and Doyle, K. (1997). Fabrik: A Visual Programming Environment, in *Proceedings of OOPSLA '88,* ACM Press, pp. 176-189.

Jaccheri, M.L., Larsen, L., and Conradi, R. (1992). Software Process Modeling and Evolution in EPOS, in *Proc. Fourth International Conference on Software Engineering and Knowledge Engineering ,* Capri, Italy, June, pp. 17-29.

Kaplan, S.M., Tolone, W.J., Bogia, D.P., and Bignoli, C. (1992). Flexible, Active Support for Collaborative Work with ConversationBuilder, in, 19*92 ACM Conference on Computer-Supported Cooperative Work,* ACM Press, 1992, pp. 378-385.

Krishnamurthy, B. and Hill, M. (1994). CSCW'94 Workshop to Explore Relationships between Research in Computer Supported Cooperative Work & Software Process, in *Proceedings of CSCW'94,* ACM Press, April, 1994, pp. 34-35.

Lonchamp, J. (1995). CPCE: A Kernel for Building Flexible Collaborative Process-Centred Environments, in *Proceedings of the 7th Conference on Software Engineering Environments,* IEEE CS Press, 1995, pp. 95-105.

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

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

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

Peuschel, B., Schäfer, W., and Wolf, S. (1992). A knowledge-based software development environment supporting cooperative work, *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 1, 76-106.

Roseman, M. and Greenberg, S. (1996). Building Real Time Groupware with GroupKit, A Groupware Toolkit, *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, 1-37, March, 1996.

Swenson, K.D. (1993). A Visual Language to Describe Collaborative Work, in *Proceedings of the, 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, pp. 298-303.

Swenson, K.D., Maxwell, R.J., Matsumoto, T., Saghari, B., and Irwin, K. (1994). A Business Process Environment Supporting Collaborative Planning, *Journal of Collaborative Computing*, vol. 1, no. 1, 1994.

TeamWARE Inc. (1996).*TeamWARE Flow*, (http://www.teamware.us.com/products/flow/).

Valetto, G. and Kaiser, G.E. (1995). Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments, in *IEEE Seventh International Workshop on Computer-Aided Software Engineering,* July, 1995, pp. 40-48.