Distributed Component Engineering using a Decentralised, Internet-based Environment

John Grundy

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

Engineering component-based software systems in a distributed fashion is challenging. Particular issues to address include software process and work co-ordination, sharing and collaborative editing of component specifications, designs and implementations, and appropriate sharing of reusable components. This paper describes our approach to tool support for distributed component engineering. Several tools are integrated for each developer, and a decentralised, multi-user work environment established. Key characteristics of this environment include flexible process management and process-based tool co-ordination, collaborative editing support, distributed work co-ordination and 3rd party tool integration agents, and distributed component storage and retrieval support. We illustrate some of these tools in use during multi-user development, and discuss the architectural realisation of these tools and their integration.

1. Introduction

Component-based systems engineering shares many commonalities with traditional software systems engineering, and uses many common modelling and implementation approaches and tools. However, it brings some new challenges as well as preserving some common problems relating to managing distributed software development projects [36, 37, 19, 40]. These include the use of new, often highly dynamic and weakly defined software processes, the need to heavily leverage existing software component designs and implementations in a consistent way, and while it often allows developers to work more independently, at various times still requires developers to tightly collaborate and co-ordinate their work.

We have been working for some time on developing flexible, open, internet-based software engineering environments [14, 19]. Recently we have been developing tools to help support large-scale component-based systems engineering, which themselves are complex component-based applications [13, 19, 20]. Assembling these tools into a component engineering environment enables multiple, potentially widely distributed developers to incrementally build and assemble complex applications using component technology. Such an environment needs to provide a range of capabilities to component engineers. These include:

- *Process modelling and enactment.* Software process management is necessary to both guide developers and track the work they have done by in a geographically and temporally distributed fashion. Even developers working together locally can greatly benefit from such support to better plan, manage and co-ordinate their work [1, 5, 14].
- *Collaborative editing.* Developers need to exchange software development artifacts and collaboratively edit these in various ways, ranging from fully synchronous to asynchronous editing [6, 13]. They also need support to version artifacts and manage component-based systems configurations made up from multiplely versioned artifacts [13, 29].
- *Distributed work co-ordination and tool integration.* Developers need to co-ordinate their tool usage with their personal software process i.e. have local tool integration and process co-ordination support. They also need the assistance of “software agents” which help them effectively manage distributed work and tool co-ordination [1, 5, 12].
- *Component management.* Component-based systems leverage existing software artifacts (components) heavily. A shared repository of such artifacts is necessary to facilitate distributed component-based systems development [20, 24, 30].

We describe the development of a component engineering environment made up of several tools: a component analysis, design and implementation tool; a software process management tool; a software architecture modelling tool; a component implementation and testing tool; and various 3rd party tools integrated by the use of component technology. We focus on the internet-based support features of these tools, particularly the recent work we have done enhancing their distributed usage and tool integration facilities. These include the integrated process support, collaborative editing, distributed work co-ordination and component repository facilities. We briefly summarise our experiences to date in developing and deploying these tools, presenting an improved model for a distributed tool integration architecture.

2. Related Work

A wide variety of work has been done in building Internet-based software engineering tools and environments. Some of the earliest approaches focused on asynchronous work with version control and configuration management facilities provided by tools like SCCS and RCS, across networked file systems. Tools like FIELD [31] and DEC FUSE [23] extended this paradigm to support some aspects of synchronous or semi-synchronous work, with message-passing used to support tool integration and user interface wrappers to achieve the appearance of a single environment. Such approaches tended to work reasonably well, though rather restrictive limitations were imposed on the degree of synchronous support offered, the extensibility of the environments and the degree of process co-ordination support possible [15].

Another early approach was to develop synchronous editors for design-level or even code-level collaborative work support. Examples included Mercury [25], ConversationBuilder [27], and COAST [35]. Many of these systems didn't originally work over internet-based communication mechanisms, although recent examples do [2]. Various toolkits to support building such applications have been developed, including GroupKit [34] and Suite [6]. Unfortunately most collaborative editing tools and toolkits lack adequate support for process management and work co-ordination, and often focus heavily on synchronous work whereas most software development activity tends to be asychronous.

Process-centred environments aim to guide or enforce the use of a codified software process to control software development. Examples include EPOS [5], SPADE [1], Serendipity-II [14] and Oz [2]. Many of these environments do not integrate well with other tools for performing collaborative software development, however, though some have had collaborative editing support [1, 3] and tool integration support [39] added.

Shared software artifact repositories have been important in supporting primarily asynchronous work. These are also needed in order to support component-based software engineering where a large library of reusable components need to be centralised for shared retrieval. Examples of such systems include document-centric ones like and BSCW [4], and software component libraries like CodeFinder [24], and that of Pai and Bai [30]. One of the main problems we have found with many of these existing approaches is their reliance on either too simple indexing strategies, or use of very complex and restrictive formal program semantics [20, 24]. Integration of a component repository with other development tools and a consistent, agreed mechanism to index and retrieve components are further challenges [20].

Various systems have taken an integrated approach to supporting distributed software development, where one tool supports a wide range of collaborative work capabilities, or multiple tools are integrated to provide a sophisticated environment. Those adopting the former approach include Argo/UML [32], Serendipity-II/JComposer [19], MOOSE [10], and TeamWave [33]. Others providing an architecture into which other tools can be integrated using flexible mechanisms include Oz [26, 39], CoCoaDoc [38], and the BA environment [9]. Many of these approaches result in either monolithic, difficult to maintain and extend environments, or limit the kinds of tools that can be added into an environment. CHIME [] takes an alternative approach to integration by providing developers with integrated browsing and hyperlinking facilities to software aritifacts, but leaves artifacts under the control and locality of the tools that modify them [7].

Various toolkits and meta-CASE tools have been developed to assist tool developers and integrators in developing distributed software development environments. These include Xanth [26], COAST [35], CocoaDoc [38], MetaEDIT+ [28], MetaMOOSE [11], and JViews [19], as well as more general-purpose technologies such as CORBA [8]. Most of these approaches focus on a limited domain of software development, such as COAST and CocoaDoc on document editing. Others such as MetaMOOSE, Xanth and JViews provide general-purpose infrastructures, but may require tool developer considerable effort to achieve a desired degree of tool integration. Limitations of their architectural approaches may also restrict the ways tool developers can achieve collaborative work support, data persistency and distribution, process co-ordination and tool integration.

3. Overview of Our Component Engineering Environment

We have developed a distributed component engineering environment by integrating several tools, both our own and 3rd party. Figure 1 illustrates this environment. The main tools comprising this environment are outlined below:

- *Process management*. Software process management is necessary to both guide developers and track the work they have done by in a geographically and temporally distributed fashion. We use our Serendipity-II process management environment to provide distributed process modelling and enactment support, along with user-programmable and deployable task automation and tool integration agents [14].
- *Software Architecture design*. The architectures of complex component-based systems require careful design and analysis. SoftArch provides software architecture modelling and analysis facilities, including reusable software architecture templates and analysis agents [22].
- *Component design*. Software component designs must be captured, and suitable component implementations realised. Our JComposer CASE tool provides component specification, design and basic code generation support [19].
- *Component implementation*. Component implementation and low-level debugging must be supported, along with higher-level component visualisation support. We allow developers to use their own preferred development tools, such as JDK, Forte or JBuilder. Our component visualisation tool, JVisualise, can be used to view running

components, and we have added high-level dynamic architecture analysis tools to SoftArch to support dynamic system visualisation [19, 22].

- *Component storage and retrieval.* A component repository allows multiple developers to share components in JComposer, and also supports the sharing Serendipity-II software agent components [20].
- *Other applications.* We have integrated these tools with Argo/UML [32] via XML-based import/export mechanisms [22], several desktop applications (including Word™ and Netscape™), and a file server (a simple form of RCS) [14].

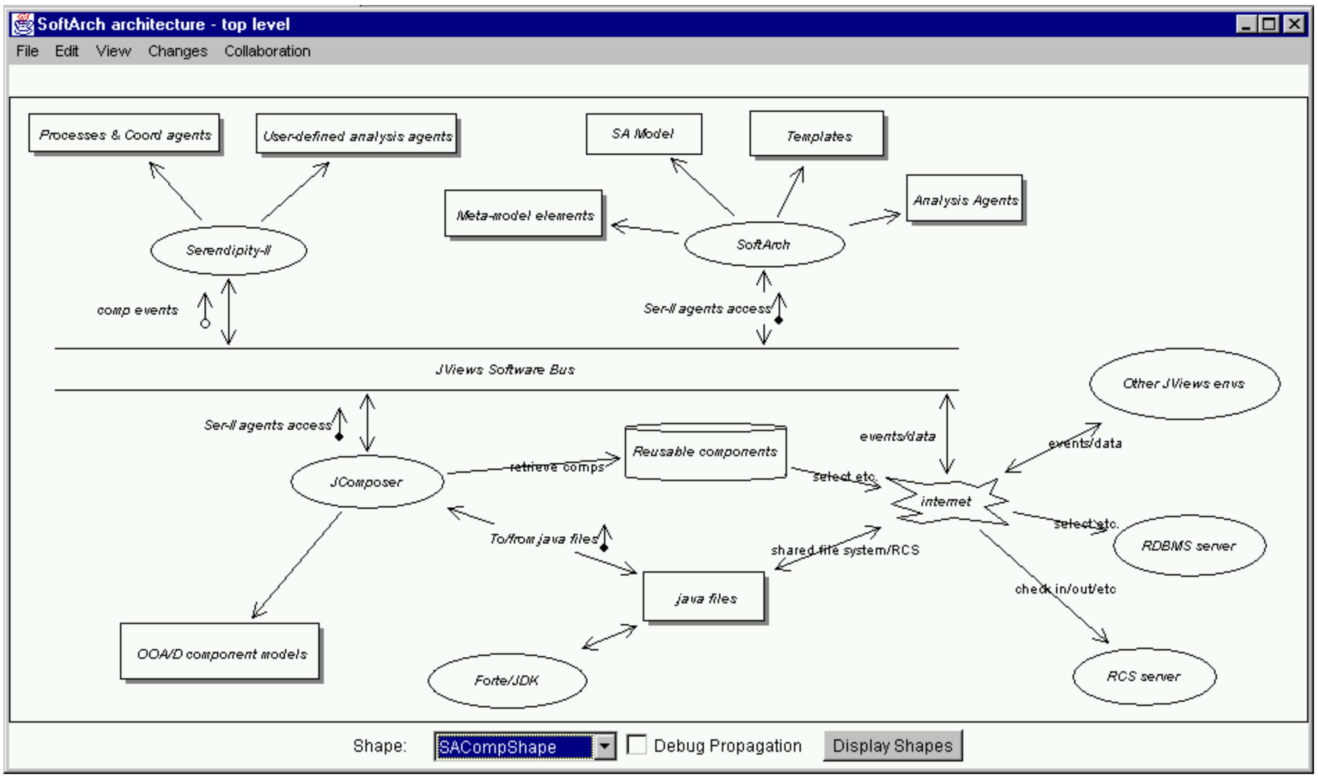


Figure 1. The basic architecture of our component engineering environment.

The basic approach we have taken to developing an integrated, distributed component management environment is shown in Figure 2. This illustrates the architectures of our decentralised software engineering environments, built using our JViews component-based framework [19]. JViews provides abstractions for building tools supporting multiple views, component-based integration mechanisms, local persistency, and distributed event and data exchange. JViews-implemented tools are integrated on a developer's local machine using JViews infrastructure abstractions (basically component event and data interchange mechanisms). All data used by these tools is stored locally, with other users' tools also storing data they use locally. Any shared data replicated on all machines, producing a decentralised architecture for the overall system [14, 19]. Such an architecture supports a robust software development environment, tolerant of network and machine failure, provides improved security for private work data, and scales up well to large multi-person projects.

Distributed software development is support across any internet communications medium (LAN or WAN) by event and data exchange between users' environments. This involves tools exchanging events via JViews event distribution abstractions to reconcile updated data, notify of interesting events etc. Data can be exchanged with a version control and configuration management mechanism used to manage alternate versions [13, 19]. Local and remote 3rd party tool access is support by wrapping tools (e.g. desktop applications like Word™ and Netcape™, or servers like RCS and http) with JViews components. Local and remote "software agents" provide both tool integration and task automation support. These allow the behaviour and composition of an environment to be extended and configured by developers as they require [14, 16, 19].

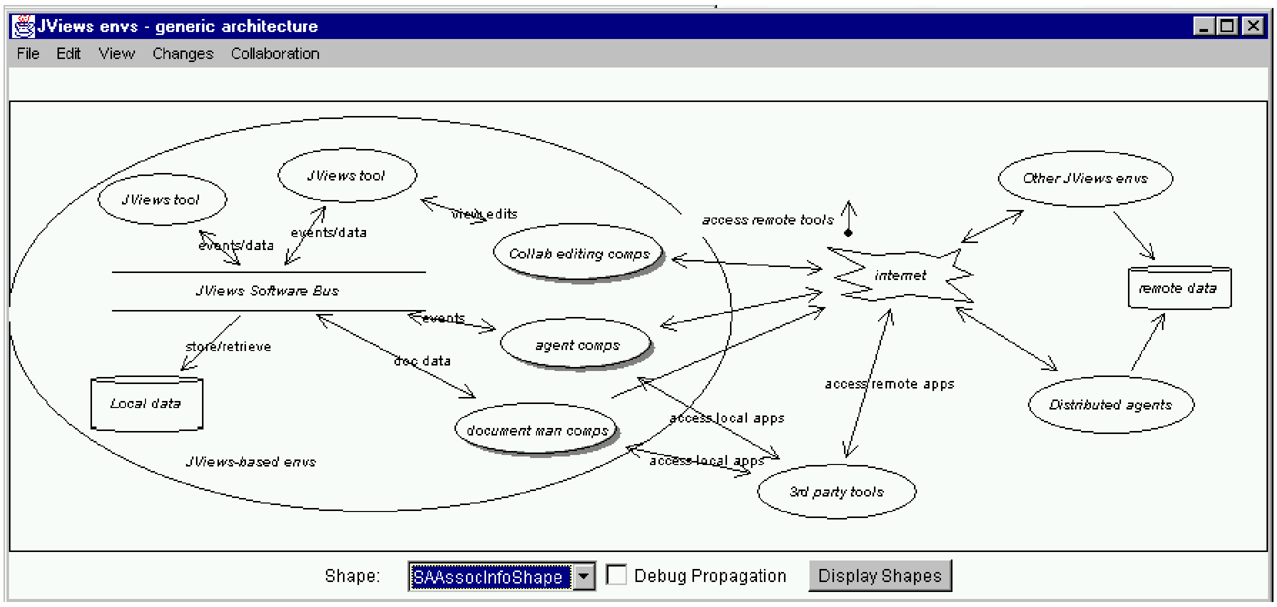
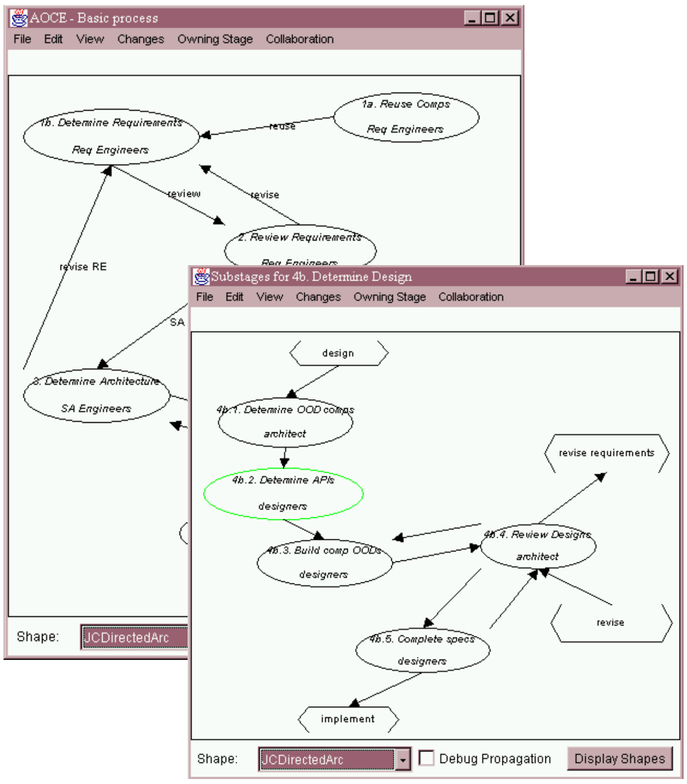


Figure 2. Basic architecture of our decentralised environments.

Our component engineering environment is supported by collaborative editing facilities plugged into our JViews-based tools, decentralised work and process co-ordination tools built and deployed in Serendipity-II, a (currently centralised) component repository, and a (currently centralised) distributed document (file) management tool. In the following sections we illustrate the process management, collaborative editing, distributed work co-ordination and distributed component management facilities provided by this environment. We briefly discuss the architectures and implementation of these facilities, particularly focusing on recent work we have done in enhancing these. We conclude with an overview of possible future research directions in this area.

4. Process Management



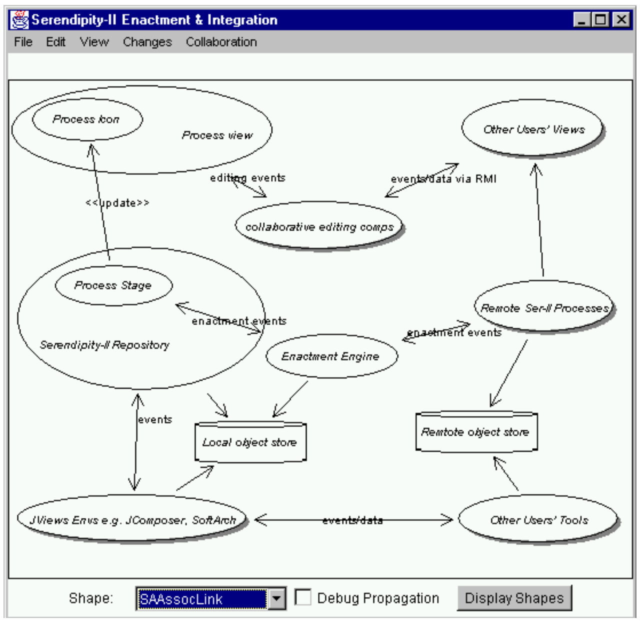


Figure 3. (a) Example of enacted process views and (b) Serendipity-II architecture.

Figure 3 shows an example of a simple component engineering process (top view) and one of the process stages expanded (bottom view) in Serendipity-II. This is part of the Aspect-oriented Component Engineering (AOCE) methodology we have been developing [20]. Serendipity-II provides multiple, visual views of software processes and project plans which can be collaboratively edited. Process stages are enacted by each developer collaborating on a project, with automation of enactment as stages are completed or by software agents monitoring work done in development tools. Serendipity-II can monitor software artifact editing being carried out in tools are store these events (or synthesised summaries of multiple events) against process stages, providing a work tracking mechanism.

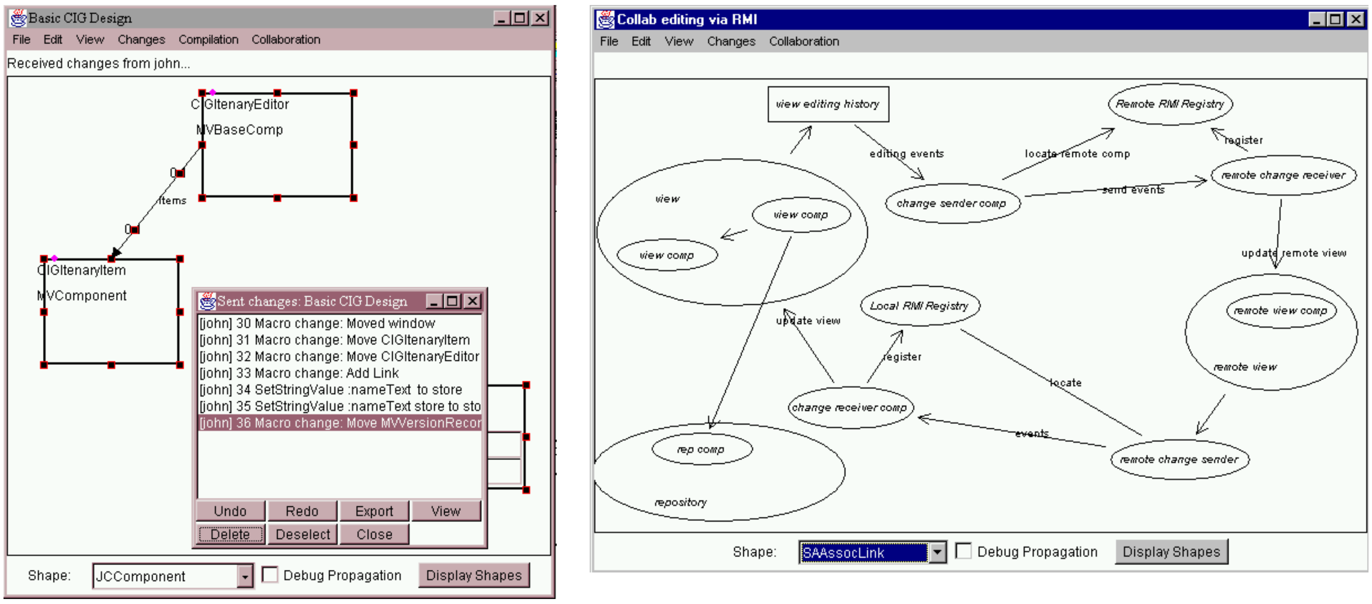
We originally built Serendipity-II's collaborative enactment, editing and distributed tool monitoring mechanisms using custom socket-based protocols. Process model and enactment state information was stored by a custom component persistency mechanism [14, 19]. This approach worked reasonably well, but suffered from performance, scalability and difficultly in coding for developers. Recently we have modified this mechanism to utilise a local object store (PSE Pro™) to manage local environment data storage. This provides a much improved mechanism for tool developers to use, and provides users much better performance and scalability, particularly when large process models are in use. Exchange of editing events and enactment events is now achieved by using the Java Remote Method Invocation API. This also is a much easier mechanism for tool developers to use, and provides a more efficient, scalable and robust data communications mechanism. Keeping process model and enactment engine information cached locally allows Serendipity-II to tolerate network or remote host failure, with developers still able to continue work. Serendipity-II manages reconciliation of enactment events when network connection is re-established.

Co-ordination of local tools by the use of Serendipity-II is achieved using a component-based event monitoring facility provided by JViews [14, 18]. Non-JViews implemented tools are wrapped by a JViews component which provides control over the tool to the degree that can be achieved by the developer given the 3rd party tool's provided interface mechanisms.

5. Collaboratively Editing Views

Developers collaboratively edit views in various tools in our environment, ranging from Serendipity-II process models, to SoftArch and JComposer architecture, analysis and design-level views, to JDK .java source code files. The only support we provide for source code file editing is asynchronous editing of different versions of the source code, checked out of a RCS-based distributed file management server [18]. This has proved adequate, as developers almost always edit implementation code independently. This is particularly true for component-based systems, where use of a component is ideally always via a well-defined interface specification, characterised at component design-time.

Higher-level views of software development, such as process models, architectures and designs, sometimes need to be more tightly shared and collaboratively edited. Our JViews-based tools provide a flexible collaborative editing mechanism that allows developers to share views and edit them synchronously or asynchronously [13, 19]. For example, Figure 4 shows a JComposer component design view being semi-synchronously edited. Changes made by one user, 'john', are displayed as they are made but not immediately applied to the view (as with synchronous editing). The user whose view this is will at some stage select one or more changes and then ask JComposer to apply them to his/her view incrementally. Inconsistencies, such as syntactic or semantic errors that occur, generate error messages that are displayed in another dialogue. The user can also edit such views asynchronously, and have JComposer send them to collaborating developers at some later stage, where they can be incorporated or further discussed/refined. Whole views can also be exhanged, creating alternative versions for developers to edit and merge further changes with [13].



Received changes from john...

```
[john] 30 Macro change: Moved window
[john] 31 Macro change: Move CIGItenaryItem
[john] 32 Macro change: Move CIGItenaryEditor
[john] 33 Macro change: Add Link
[john] 34 SetStringValue :nameText  to store
[john] 35 SetStringValue :nameText store to sto
[john] 36 Macro change: Move MVVersionRecor
```

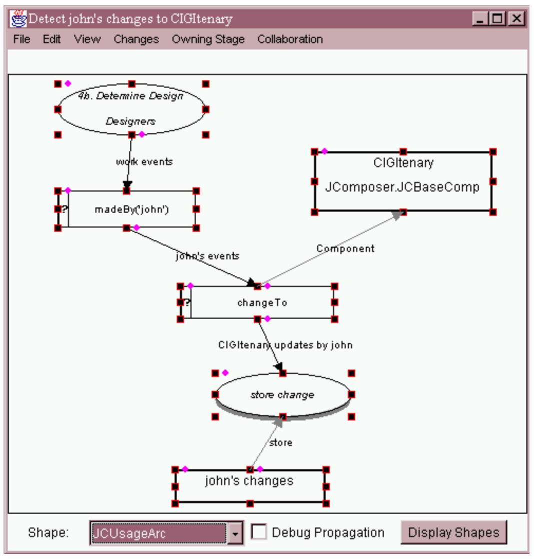
Figure 4. (a) Example of collaborative editing in JComposer and (b) its architecture.

We originally implemented our collaborative editing mechanism using custom socket-based event and data exchange [19, 16]. Due to problems identified in the previous section, we recently replaced with this with an RMI-based mechanism, to provide a more robust, faster and easier to program implementation. Figure 4 (b) illustrates the architecture of the JViews components implementing this collaborative editing mechanism. Any JViews-based environment can have these components plugged into it (dynamically), and will then support this flexible collaborative editing mechanism over any internet-capable communications link.

Each JViews environment maintains event and data sender/receiver components. It also has a Java RMI registry, with receivers registering themselves for remote object access. When sending events or view component data, the sender component contacts the RMI registry on the collaborating user(s) host and locates the remote recveiver(s) to send the events/data to. When receiving the events/data, the receiver component maps object IDs to local object IDs (recall all shared data is replicated) using a JViews-provided object registry, and merges changes into the remote view or presents to the other user(s). When other users edit a shared view, changes are sent to a user’s change receiver, which presents or merges changes appropriately. Our collaborative editing mechanism is tolerant to network failure, with events marked so they can be resent when a network connection is re-established. Users can continue to asynchronously edit any shared view if a connection fails, then merge and/or review changes made by others, illustrating the advantage of a decentralised architecture.

6. Distributing Task Automation Agents

Local tool co-ordination and local process-based task automation agents are programmed using JViews components. These are driven by events produced by JViews-based tools (or wrapped 3rd party tools) and act on local tool data. Distributed work co-ordination and tool integration is necessary when multiple developers need to be made aware of work others are doing/have been doing, or when a remote server needs to be used to centralise data or processing. For example, the Serendipity-II work co-ordination agent shown in Figure 5 (a) is an example of a task automation agent implementing a simple notification mechanism. In this example, when editing events made by user “john” occur while the process stage “4b. Determine Design ” is enacted, they are filtered to see if they apply to the component "CIGItenary", and if so are stored so user “mark” can review them later.



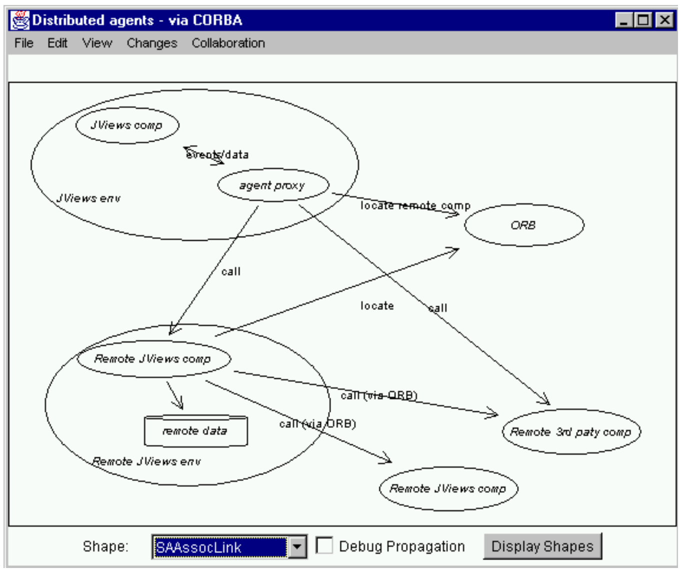


Figure 5. (a) Work co-ordination using distributed task automation agents and (b) its architecture.

Such task automation agents are programmed using a visual agent programming language in Serendipity-II, and may include components corresponding to remote tools or named software agent. Events need to be detected from these remote components and sent to the local JViews environment. Some local components making up the task automation agent specification may also need to send events/operation invocation messages to remote tools and/or agents. Security must be maintained, ensuring task automation agents can not corrupt remote data or incorrectly invoke remote tool and agent operations. Similarly, the decentralised nature of our overall environment is something we wish to preserve, meaning agents must be built tolerant to the possibility of network failure occuring. Any remote data ideally should be replicated locally where possible to ensure the agent can function during network failure and to minimise network traffic.

Our original implementation of remote task automation and tool integration agents used our custom socket-based protocol and component serialisation/deserialisation mechanism built into JViews [18, 19]. This proved to be difficult to use and program, and suffered not so much from scalability and performance problems, as our originally developed collaborative editing components did, but from a lack of open systems integration. We have recently prototyped some new task automation agents using CORBA-based remote object communication. Our CORBA-based remote task automation agents use a remote CORBA ORB (again, each users host runs an ORB) to locate and invoke remote agent operations, and to request remote agents send them events. 3rd party tools and components which provide a CORBA-compliant remote object interface can be integrated without JViews wrapper components. The naming and remote operation invocation facilities of an ORB give improved performance and ease-of-development for our remote agents than our previous implementation approach.

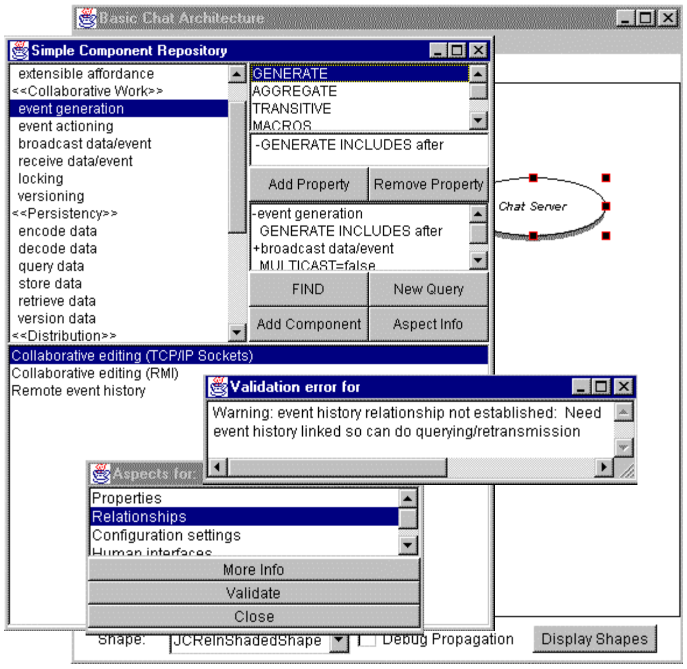
We do not use remote CORBA objects for data access, as these are problematic both in terms of performance (if a large number of remote agents need access to this data) and when network connections/hosts fail. Instead we use JViews' data replication infrastructure support and local persistency mechanism to decentralise data access whenever possible. CORBA-based operation invocation is used to support this replication, and is used as the communication medium for event broadcasting when reconciling distributed data.

7. Distributed Component Storage and Retrieval

We have recently developed a component repository (currently using a centralised, rather than decentralised data management mechanism). This allows distributed developers to share components, particularly commonly reused infrastructure components for building applications. In order for developers to effectively share components, a common language or ontology for characterising and describing components must be used. We have developed a development methodology for component engineering that we call aspect-oriented component engineering [17]. This provides, in addition to a process, design notation and implementation framework, a standardised way of characterising components. Aspects capture horizontal perspectives of a component-based system, such as user interface, persistency, distribution, collaborative work, security, component configuration etc. characteristics. They can also capture domain-specific vertical perspectives, such as process modelling and enactment, data modelling, processing and storage, and so on.

Figure 6 (a) shows an example of a developer searching our repository for a component supporting event broadcasting over the internet (from [20]). The developer formulates a query using aspect, aspect detail and aspect detail properties of the component they are searching for. Retrieved components are displayed, and aspect-organised information about these components can be browsed by the developer. In this example, the developer has reused a socket-implementing collaborative editing component (actually part of the JViews infrastructure). A validation function has been run and the component is indicating it currently lacks a required relationship to another component in order to be able to function.

The architecture of this component repository is illustrated in Figure 7 (b). Currently the component repository client uses a centralised component repository server, which access a relational database in which component information is stored. Component descriptions in the database provide component implementation and design documentation links (in our systems, components are JavaBeans, and thus have .class bytecode files implementing the component, sometimes .java source code files are available, and sometimes Word™, UML or JComposer OOD models documenting them are available). We are planning to extend this architecture to allow environments to have a local RDBMS server storing component information. A component query will then traverse the local repository first, using other users' and other distributed repositories for further component searches. Developers can add components to a repository, with JViews' aspect extensions used to automate indexing of components [20].



extensible affordance
<<Collaborative Work>>
event generation
event actioning
broadcast data/event
receive data/event
locking
versioning
<<Persistency>>
encode data
decode data
query data
store data
retrieve data
version data
<<Distribution>>

GENERATE
AGGREGATE
TRANSITIVE
MACROS

-GENERATE INCLUDES after

-event generation
GENERATE INCLUDES after
+broadcast data/event
MULTICAST=false

Collaborative editing (TCP/IP Sockets)
Collaborative editing (RMI)
Remote event history

Warning: event history relationship not established: Need event history linked so can do querying/retransmission

Properties
Relationships
Configuration settings
Human interfaces

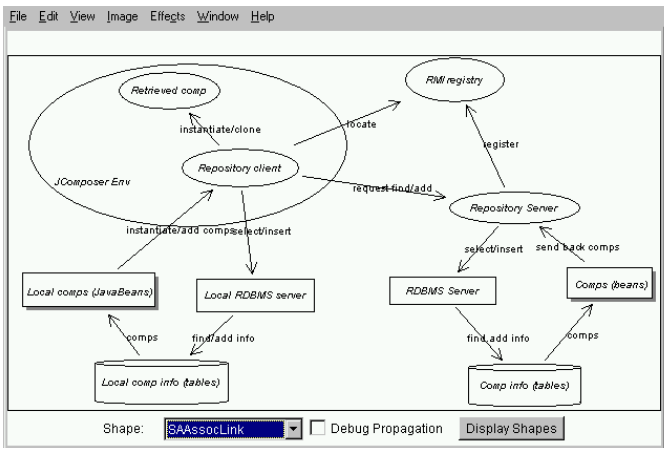


Figure 6. (a) Distributed component retrieval and (b) its architecture.

8. Future Work

To date our component engineering tools and overall environment has proved useful for various small-to-medium system development tasks [14, 19]. We are in the process of evaluating some of the recent architectural enhancements we have made, in addition to new tools like SoftArch. This is likely to indicate new tools that may be needed, or existing tools which can usefully be integrated into the environment, as well as existing tool enhancements. Some current enhancements to the environment we are planning include distributed running system visualisation and testing support in SoftArch [22], adaptable user interface component design and development [21], and improved support for aspect-oriented component engineering [17].

Our recent work has indicated that we also need to further investigate the infrastructural support for building and integrating internet-based software engineering tools. Various enhancements we have built have improved the ability of tool developers to build and deploy software tools, but have made our JViews framework rather complex and unwieldy. Developers must still be aware of local vs distributed components when developing tools, use different naming and operation invocation services for RMI and CORBA-based distributed components, and are forced to adhere to various application-specific constraints when using PSE Pro™ and RDBMS (via JDBC) persistency mechanisms for tools and the component repository. In addition, as tools must inherit or aggregate most distributed support facilities from JViews classes, this limits the ability for distributed support infrastructure to be dynamically modified, as it is “hard-coded” into tools. We would like to make the various distributed work facilities described in this paper far more transparent to tool developers as well as tool users, and allow it to me more flexible.

Figure 7 gives a high-level illustratation of our proposed improved infrastructure. Various software engineering tools, such as Serendipity-II, JComposer, SoftArch and our component repository utilise facilities provided by an event-based tool integration infrastructure (the E-bus). This infrastructure allows each tool to communicate with other tools using events, including converting operation invocation requests and returns into events. This allows much more freedom in integrating tools and accessing them across the internet, as well as modifying their behaviour and the tool composition of an environment [19, 18]. Various facilities are provided transparently by this infrastructure, with each a “horizontal tool” in its own right. This includes local persistency and distributed component replication mechanisms, local and distributed software agents (components), document management facilities (including reusable component management), collaborative editing and event broadcasting, and 3rd party tool and service integration support. This is in contrast to tools making use of framework class capabilities as in JViews, which tends to limit their degree of flexibility, extensibility and does not support evolving infrastructure facilities well.

As part of this work we plan to investigate the use of the new Jini distributed component support infrastructure for Java. We plan to use our component aspect characterisation approach to characterise E-bus services for advertising and access over a Jini-based infrastructure, using event exchange. Local and remote software agents, process engines and editable views will appear the same to clients, with the E-bus and service tool components dynamically resolving local and remote component access. A unified approach to document management, including indexing and hyper-linking support, will enable us to build templates (in Serendipity-II and SoftArch), a component repository and documentation for components (including aspect characterisation) in a uniform manner. Use of PSE Pro™ for the component repository, as well as tool local persistency management, will be included as part of this work.

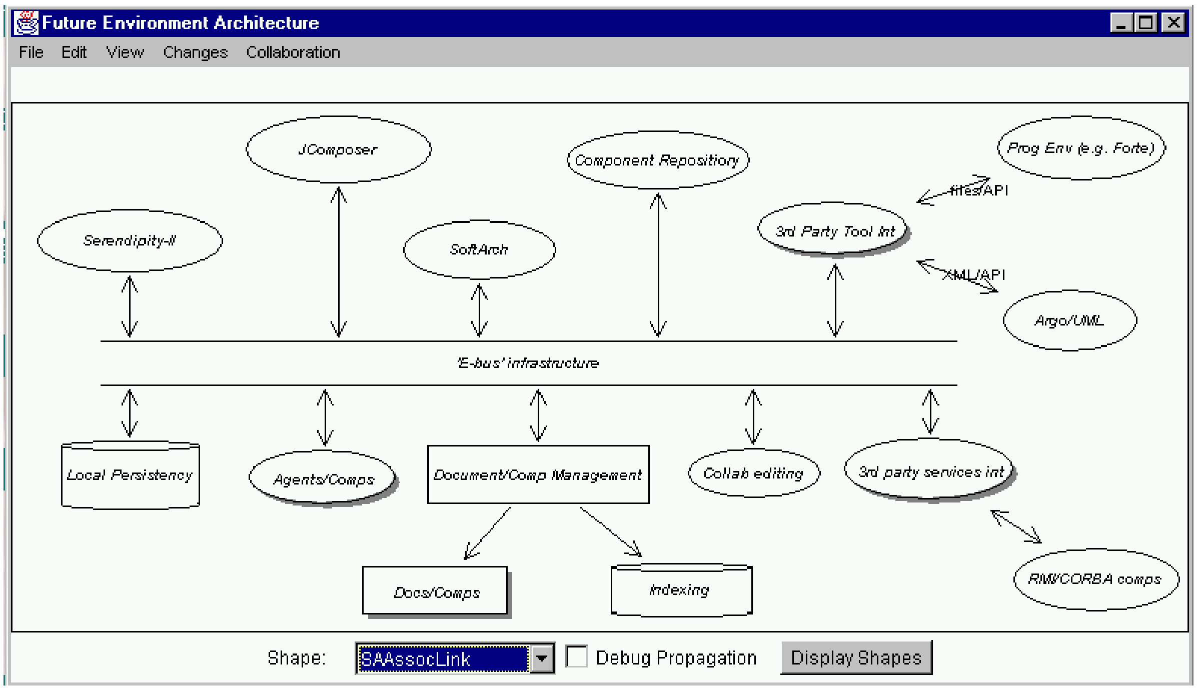


Figure 7. Future internet-based software engineering environment architecture.

9. Summary

We have developed an integrated environment for component engineering. This includes tools to support process management and local tool integration and task automation, flexible collaborative editing, distributed work co-ordination, and a distributed component repository. Some 3rd party tools have been integrated into this environment, including a programming environment and distributed file repository. These tools can all be used over the internet to support distributed software development of component-based systems. Most utilise a decentralised software architecture, which is fault-tolerant and provides good performance over a range of internet communication technologies. Further enhancements to this environment include the addition of distributed testing and monitoring tool support, distributed user interface development tools, and further work codifying the processes, notations and implementation technologies for our aspect-oriented component engineering methodology. Infrastructural enhancements are aimed at providing tool developers with a cleaner, more extensible architecture on which to build and integrate tools and tool services.

References

1. Bandinelli, S. and DiNitto, E. and Fuggetta, A., Supporting cooperation in the SPADE-1 environment, *IEEE Transactions on Software Engineering,* vol. 22, no. 12, December 1996, 841-865.
2. Begole, J., Struble, C.A., Shaffer, C.A. and Smith, R.B. Transparent Sharing of Java Applets: A Replicated Approach. In *Proc. ACM UIST '97,* pages 55-64. ACM Press, 1997.
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,* IEEE CS Press, Airlie, VA, October 1994, pp. 114-116.
4. Bentley, R., Horstmann, T., Sikkel, K., and Trevor, J. Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system, in *Proceedings of the 4th International WWW Conference,* Boston, MA, December 1995.
5. Conradi, R. Hagaseth, M., Larsen, J., Nguyen, M.N., Munch, B.P., Westby, P.H., Zhu, W. and Jaccheri, M.L., EPOS: Object Oriented Coopeartive Process Modeling, In *Software Process Modeling & Technology*, Finkelstein, A., Kramer, J. and Nuseibeh, B. Eds, Research Studies Press, 1994.
6. Dewan, P. and Choudhary, R. A High-Level and Flexible Framework for Implementing Multiuser User Interfaces. *ACM TOIS,* vol. 10, no. 4, October 1992, ACM Press, 345-380.
7. Dossick, S.E. and Kaiser, G.E. Distributed Software Development with CHIME, In Proceedings of the 1999 ICSE Workshop on Software Engineering over the Internet, http://sern.cpsc.ucalgary.ca/~maurer/ICSE99WS/ICSE99WS.html.
8. Emmerich, W., " CORBA and ODBMSs in Viewpoint Development Environment Architectures.," in *Proceedings of the 4th International Conference on Object-Oriented Information Systems,* Springer Verlag, 1997, pp. 347-360.
9. Emmerich, W., Arlow, J., Madec, J., and Phoenix., M., "Tool Construction for the British Airways SEE with the O2 ODBMS," *Theory and Practice of Object Systems*, vol. 3, no. 3, 213-231, 1997.
10. Ferguson R.I., Parrington N.F. & Dunne P., 1994. MOOSE: A Method Designed for Ease of Maintenance. In *Proceedings of the International Conference on Quality Software Production* 1994 (ICQSP 94), Hong Kong, IFIP.
11. Ferguson, R.I., Parrington, N.F., Dunne, P. Hardy, C., Archibald, J.M. and Thompson, J.B. MetaMOOSE - an Object-Oriented Framework for the construction of CASE tools, *Information and Software Technology*, vol. 42, no. 2, January 2000.
12. Fernström, C. ProcessWEAVER: Adding process support to UNIX, In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, Berlin, Germany, February 1993, IEEE CS Press, pp. 12-26.
13. Grundy, J.C. Human Interaction Issues for User-configurable Collaborative Editing Systems, In *Proceedings of APCHI'98*, Tokyo, Japan, July 15-17 1998, IEEE CS Press, pp. 145-150.
14. Grundy, J.C., Hosking, J.G., Mugridge, W.B. and Apperley, M.D. An architecture for decentralised process modelling and enactment, *IEEE Internet Computing*, Vol. 2, No. 5, September/October 1998, IEEE CS Press.
15. Grundy, J.C., Hosking, J.G. and Mugridge, W.B. Inconsistency Management for Multiple-View Software Development Environments, IEEE Transactions on Software Engineering, vol. 24, no. 11, November 1998, IEEE CS Press, pp. 960-981.
16. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D., Tool integration, collaborative work and user interaction issues in component-based software architectures, In *Proceedings of TOOLS Pacific '98*, Melbourne, Australia, 24-26 November, IEEE CS Press.
17. Grundy, J.C. Aspect-oriented Requirements Engineering for Component-based Software Systems, In *Proceedings of the 4th IEEE Symposium on Requirements Engineering*, Limerick, Ireland, June 8-11 1999, IEEE CS Press, pp. 84-91.
18. Grundy, J.C. Visual specification and monitoring of software agents in decentralised process-centred environments, *International Journal on Software Engineering and Knowledge Engineering: Special Issue on Visual Programming for Parallel and Distributed Computing*, Vol. 9, No. 4, World Scientific Publishing Company, August 1999, pp. 425-444.
19. Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Constructing component-based software engineering environments: issues and experiences, *Information and Software Technology*, vol. 42, no. 2, January 2000, Elsevier.
20. Grundy, J.C. Component storage and retrieval using aspects, In *Proceedings of the 2000 Australian Computer Science Conference*, Canberra, Australia, Jan 31-Feb 3 2000, IEEE CS Press.
21. Grundy, J.C. and Hosking, J.G. Developing Adaptable User Interfaces for Component-based Systems, In *Proceedings of the 1st Australian User Interface Conference*, Canberra, Australia, Jan 30-Feb 3 2000, IEEE CS Press.
22. Grundy, J.C. *Software Architecture Modelling, Analysis and Implementation with SoftArch*, Technical Report, Department of Computer Science, University of Auckland, December 1999.
23. Hart, R.O. and Lupton, G., "DECFUSE: Building a graphical software development environment from Unix tools," *Digital Tech Journal*, vol. 7, no. 2, 5-19, 1995.

24. Henninger, S. Supporting the Construction and Evolution of Component Repositories, In *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany, 1996, IEEE CS Press, pp. 279-288.
25. Kaiser, G.E., Kaplan, S.M., and Micallef, J. (1987a). "Multiuser, Distributed Language-Based Environments*,"* *IEEE Software*, vol. 4, no. pp. 11, 58-67.
26. Kaiser, G.E. and Dossick, S. Workgroup middleware for distributed projects, in *Proceedings of IEEE WETICE'98*, Stanford, June 17-19 1998, IEEE CS Press, pp. 63-68.
27. Kaplan, S.M., Tolone, W.J., Bogia, D.P., and Bignoli, C., "Flexible, Active Support for Collaborative Work with ConversationBuilder," in *1992 ACM Conference on Computer-Supported Cooperative Work,* ACM Press, 1992, pp. 378-385.
28. Kelly, S., Lyytinen, K., and Rossi, M., "Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment," In *Proceedings of CAiSE'96,* Lecture Notes in Computer Science 1080, Springer-Verlag, Heraklion, Crete, Greece, May 1996, pp. 1-21.
29. Magnusson, B. and 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*, LA, 1993, pp. 7-10.
30. Pai, Y. and Bai, P. Retrieving software components by execution, In *Proeedings of the. 1st Component Users Conference,* Munich, July 14-18 1996, SIGS Books, pp. 39-48.
31. Reiss, S.P., "Connecting Tools Using Message Passing in the Field Environment," *IEEE Software*, vol. 7, no. 7, 57-66, July 1990.
32. Robbins, J. Hilbert, D.M. and Redmiles, D.F. Extending design environments to software architecture design, *Automated Software Engineering*, vol. 5, No. 3, July 1998, 261-390.
33. Roseman, M. and Greenberg, S. TeamRooms: Network Places for Collaboration. In *Proceedings of ACM CSCW 96,* pp. 325-333, 1996.
34. Roseman, M. and Greenberg, S. Building Real Time Groupware with GroupKit, A Groupware Toolkit, *ACM Transactions on Computer-Human Interaction*, vol 3, no. 1, 1-37, March 1996.
35. Shuckman, C., Kirchner, L., Schummer, J. and Haake, J.M. Designing object-oriented synchronous groupware with COAST, in *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, ACM Press, November 1996, pp. 21-29.
36. Sessions, R. *COM and DCOM: Microsoft's vision for distributed objects*, John Wiley & Sons 1998.
37. Szyperski, C.A. *Component Software: Beyond Object-oriented Programming*, Addison-Wesley, 1997.
38. t er Hofte, G.H. and H.J. van der Lugt, 'CoCoDoc : A framework for collaborative compound document editing based on OpenDoc and CORBA'. In *Proceedings of the IFIP/IEEE international conference on open distributed processing and distributed platforms*, Toronto, Canada, May 26-30, 1997. Chapman & Hall, London, 1997, p. 15-33.
39. 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.
40. Weyuker, E.J. Testing Component-based Sotware: A Cautionary Tale, *IEEE Software*, Sept/Oct 1998, pp. 54-59.