Supporting dynamic software tool integration via web service-based components

Nelson Yap[1], Hau Chean Chiong[1], John Grundy[1, 2] and Rebecca Berrigan[2]
Department of Electrical and Computer Engineering[1] and Department of Computer Science[2], University of Auckland, Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

Most software engineering tools come with fixed functionality or limited plug-in extension capabilities. Building software development environments that support truly dynamic extension capabilities to incorporate a wide range of additional facilities at run-time has proved to be a very challenging task. We describe a new approach using web service components to support the dynamic discovery, integration and invocation of remote software tool facilities for JEdit, an open source Integrated Development Environment. In this approach discrete software tool functionality is encapsulated in software "toolets", accessed as remote web service-based components. These toolet services are registered and discovered, and then dynamically integrated and invoked from within the JEdit IDE as required. We describe the architecture of our approach, key design and implementation issues, and illustrate the feasibility of the approach with several prototype toolet components and results of evaluations.

Keywords: software tool integration, web services, service discovery and integration, integrated development environments

1. Introduction

A single software engineering tool seldom provides a complete set of facilities for all of its potential users. Thus software tool integration has become a major research and practical area of work in software engineering, allowing developers to compose an environment from multiple, integrated tools [18, 22, 25].

Common approaches to software tool integration include data integration via common file formats or shared data repositories [8, 26]; control integration via event passing, plug-in APIs or remote object APIs for tools [12, 22, 28]; presentation integration via plug-ins and wrappers [22, 23, 15]; and process integration via workflow tools and process-centred environments [1, 2, 3]. All of these approaches trade off different advantages and disadvantages. Common problems encountered include a need to install tools on all potential user machines and keep up-to-date; difficultly in tightly integrating client PC IDEs with distributed server facilities; and the complexity and incompatibility between different plug-in and remote object APIs for software tools.

To try and overcome these problems we have developed a proof-of-concept approach to supporting dynamic software tool extension via web service-based remote software components. In this work we extended an open-source Integrated Development Environment (IDE), JEdit, to support dynamic discovery of new tool facilities via web service component registries. These newly discovered software tool components, which we call "toolets", are integrated into this extended "JEdit-WS" IDE and invoked remotely via XML web service messages. The remotely hosted toolet services process requests from multiple users and results are displayed within the client JEdit-WS environments. To demonstrate the feasibility of this concept we have developed remote services for version control, collaborative messaging, code refactoring and code inspection. All of these use remote web service component interfaces to existing third-party software tools to provide these facilities. Web service wrappers were used to provide data and control integration with these remote web services, with JEdit-WS providing a consistent presentation integration strategy for them.

We firstly provide a motivation for this work and summarise key contributions to date of related research. We then outline our approach of using a remote web service component-based tool extension mechanism. We illustrate our approach with several toolet examples used to dynamically extend our proof-of-concept JEdit-WS IDE. We describe key design and implementation issues and provide an evaluation of our approach's key strengths and weaknesses. We conclude with a summary of the contributions of our work and directions for future research.

2. Motivation

Integrated Development Environments (IDEs), by their very name, provide software developers with a way of integrating the toolsets that improves their personal productivity and code quality. From an organisational software production perspective, software vendors are keen to benefit from this increased productivity and quality that a good IDE promises to deliver.

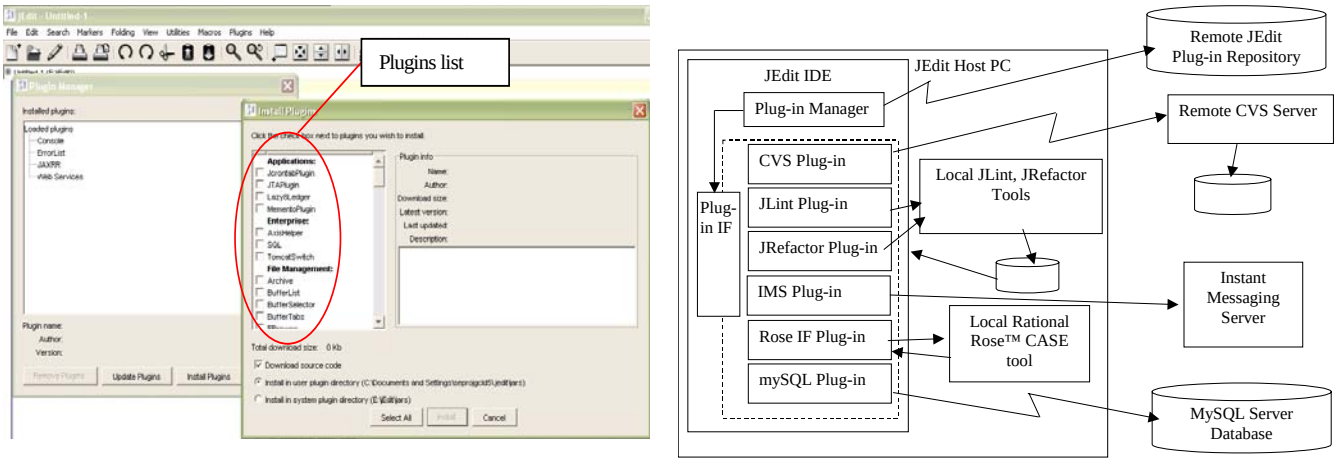

Figure 1. (a) Example of a dynamically extensible JEdit IDE and (b) its possible software architecture.

However, several factors can limit the suitability of an IDE for use within a software production organisation.

Diversity : Software developers, through training and diverse experience, will each have individual preferences regarding the tools that suit them the best in their daily work habits. In recent times this has started to be catered for through IDE plug-in solutions where toolsets can be composed by the developer using a framework for integration. Many new Integrated Development Environments (IDEs) and CASE tools, such as Visual Studio™, Eclipse, Rational Rose™, Together™, Visio™ and JBuilder™, provide a large range of facilities for developers, even if many are never used or are under-used. In addition, many provide plug-in based extension mechanisms allowing new facilities to be added by a software component plug-in approach. Each user wanting a new tool facility uses a plug-in to their IDE or CASE tool to access the facility. This approach typically requires copies of the new tool facility held on each developer's PC and sometimes very detailed knowledge of the IDE's plug-in approach to integrate it.

Figure 1 (a) shows an example of a typical IDE, JEdit [13]. New plug-in facilities can be integrated within the JEdit IDE by use of a plug-in manager, with each plug-in requiring a wrapper interface implemented following a standard convention. The architecture of such a tool is illustrated in Figure 1 (b). The IDE "client" is extended by plug-in components conforming to this interface standard and which typically access tool facilities implemented in the same language as the IDE (Java in the case of JEdit). Some remote tool services e.g. a version control server, may be accessed by a custom protocol with the plug-in providing an adaptor to this. Third-party tools may be accessed in the same manner. While a range of extensions may be supported dynamically via plug-in components, many kinds of tool extension typically require modification of the tool itself e.g. collaborative editing, tool editor enhancements, code generation and reverse engineering support.

Such approaches to tool integration are limited in that they require each version of an IDE to be extended, sometimes manually, with copies of the plug-ins providing the new tool facilities distributed. Despite the flexibility of the plug-in solution, the plug-ins must be written specifically for designated IDEs, promoting in some cases IDE lock-in where, contrary to intentions, the choice of plug-ins will dictate the preferred IDE.

Many tool integration approaches are not dynamic and require tool modification or at the least advanced user support for enhancement. As copies of plug-ins are distributed developers may not be using the latest and best versions of tool extensions. If a distributed extension isn't actually used it may waste space and still cost the organisation for purchase and upgrade. Considerable knowledge of a tool architecture is typically required to find and deploy extensions.

Opportunistic vs. Mandated Practice : Within the suite of tools employed by an organisation, some provide best organisational gains through mandated use. These tools include time logging, source control, project management, change request maintenance. While the business case for purchase of mandated tools is clear, the costs associated with integration with IDEs is often underestimated and will tend to lock-in a particular IDE, thwarting the productivity through diversity approach referred to above.

Tool use in development is not always mandated. Opportunistic tool use can be common, where developers use tools sporadically as a succinct service e.g. they supply code or designs for processing and receive a report (which is not recorded, stored or managed). The business case for installation and/or upgrading of such facilities is not so strong where tools are not regularly utilised, as usage is

unpredictable so committing to large numbers of software licenses is not always warranted.

Process : In both cases where tools are used through either personal preference or an orgnisational mandate, we know that tool selection can affect and be influenced by an organisation's software development process. These affects are not easy to measure as time spent using tools as a part of process is not generally measured. In general, to impinge upon an organisation's day to day environment with experiments that have real world consequences is a questionable strategy. There is merit, however, in monitoring usage of a range of different IDEs in conjunction with a selection of tools. This provides invaluable data that is not generally available to organisations with respect to committing to tool purchase. However, such monitoring is very difficult where tools and plug-ins are all installed locally and where the plug-in mechanisms don't support such monitoring activities.

Licensing Policy : IDE vendors and standard tool vendors alike offer single user trial licenses for short periods in order for a user to evaluate the tool or IDE. This evaluation is intended for the user to check basic usability and to evaluate feature sets in order to decide whether to purchase. Tool evaluation rarely involves the tool in high risk work, as very few tool consumers will be comfortable with the use of a tool as a dependent part of a development system knowing that licensing will expire. The ability for a range of developers to use a tool on a pay per use basis is attractive to business as the business case for purchase becomes clear over time and useful work is still facilitated. A centrally co-ordinated service to facilitate the integration of a range of business users with a range of tool-based services and manage micropayment would serve to address a tangible overhead in evaluation costs.

3. Related Work

A range of integration approaches for software tools have been developed over many years [10, 18, 27]. Most integration approaches tend to focus on supporting data integration [8, 26], control integration [22, 23], user interface integration [15, 22], and/or process integration [1, 2]. Many are not dynamic, requiring a toolset to be manually extended by code or script development, and typically require each environment to be extended rather than using shared tool facilities. Dynamically extending software engineering environments, i.e. extending a toolset while it is in use, is an especially challenging task. Limited work has been done in this area, generally focusing on supporting a limited range of tool plug-ins or data-based information exchanges between tools.

Data-oriented tool integration approaches have included CASE tool data exchange [26], shared object-oriented databases [8], special-purpose tool databases like PCTE and active repositories [25, 17], and federated systems [3]. The main disadvantages of these approaches are lack of dynamic integration support and limitation to data-oriented exchanges. User interface integration techniques provide a common user interface metaphor e.g. GUI wrappers [22], a common interface library [5], or WWW [15], but often lack back-end integration support for the tools [23]. Remote object-based integration approaches include use of CORBA and related distributed object technologies [8], software components [11, 12, 28], and web services, particularly for workflow system integration [3, 14, 20]. These approaches provide powerful integration support, but often lack adequate user interface and process integration support across the integrated tool sets [18, 11, 23]. Process-based integration approaches typically use either data integration or remote object services to achieve process-based co-ordination of tool usage [12, 1, 2, 3]. They often also achieve a limited degree of user interface integration via the process co-ordination tool interface e.g. shared to-do lists. Such approaches to date have been limited by the degree of data and control integration mechanisms provided by the tools to integrate, resulting in data redundancy and inconsistency and limited ability to invoke other tool facilities [1]. Meta-tools are an alternative approach, where users can build and tailor tools to their own needs dynamically [7, 9, 16]. However these approaches often take great effort and most meta-tools lack integration support for existing third-party tools.

We believe that Web Services could provide a powerful integration technology for facilitating a variety of IDE extensions within a software development organisation, both for mandated tools and opportunistic tool usage. Web service-based integration can support data exchange (via XML documents), control-oriented co-ordination (via SOAP messages), a degree of user interface integration (via user interface synthesis by clients), and process integration (via web service-based workflow specification and enactment). Dynamic discovery of web services during IDE usage supports incremental rather than fixed integration mechanisms [20, 19]. Web services support flexible tool integration using either data, control or process-oriented facilities as appropriate, lack of distributed tool installation and upgrading, tool usage monitoring, and flexible (pay per use) licensing.

4. Overview of Our Approach

We wanted to investigate a new approach to supporting dynamic software tool integration and extension using web service component technologies. The aim of this work was to allow an IDE or CASE tool to be dynamically extended with new facilities while in use.

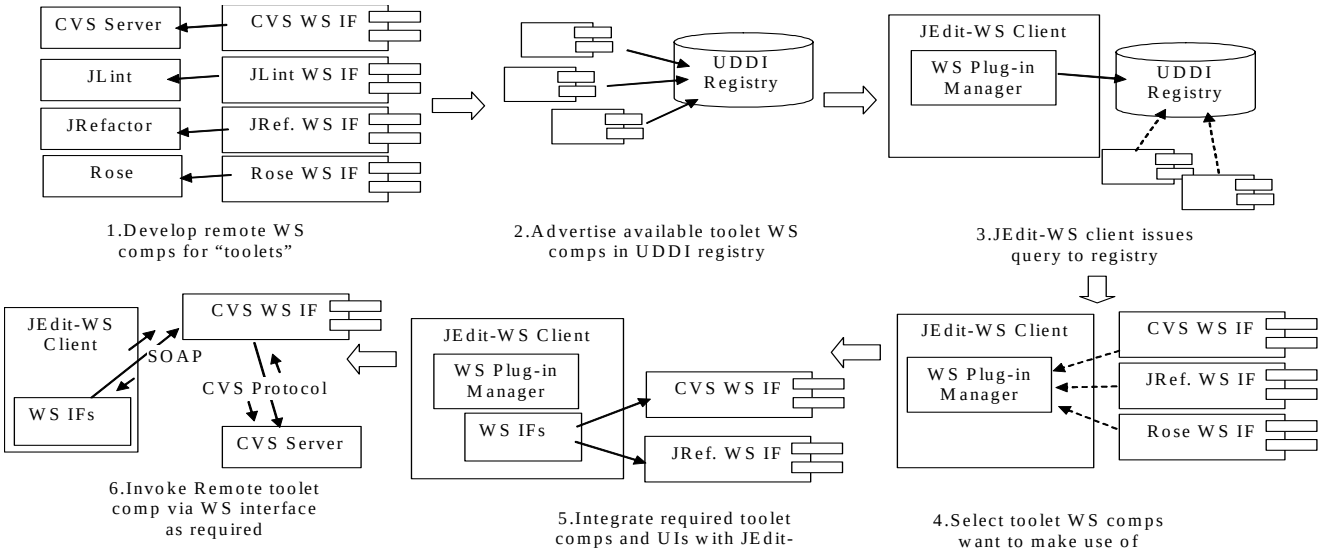

Figure 2. Overview of our approach.

Wherever possible it would make use of remote "toolet" facilities via web service component interfaces rather than copy toolets locally. Web services provide a remote object invocation protocol using standard internet transport (HTTP) and representation (XML) technologies, along with a remote server description and reflection mechanism (Web Service Description Language, WSDL) and registry system (Universal Discovery, Description and Integration, UDDI) for dynamically locating remote component interfaces [19].

Figure 2 shows how our approach works for our extended JEdit-WS prototype IDE. Various tool extension facilities are enabled for use by the development of web service interfaces and WSDL descriptions (1). Examples might include CVS (version control), JRefactor (code refactoring tool), JLint (code quality assessment tool), Rational Rose™ CASE tool interface (for code generation and reverse engineering support), Instant Messaging Service (for collaboration), and so on. These web service interfaces are advertised in a UDDI registry (2), enabling a client IDE application to locate them dynamically (3). The user may be presented with options for remote toolets to integrate into their IDE, or the IDE may use user preferences to select from located toolet services automatically (4). Software interfaces are synthesised to enable communication with the remote toolet web service and user interfaces synthesised for IDE user interaction with the remote toolet (5). As required, the IDE invokes the remote toolet via its web service interface, results being returned and processed by the IDE (6).

5. Architecture

We developed a new JEdit plug-in for locating, integrating and accessing remote toolets. The architecture of our approach is illustrated in Figure 3. A user wishing to make use of web service-based extensions to JEdit installs our WS toolet plug-in (1). The WS toolet plug-in requests available remote toolet services from one (or more) UDDI registries (2), and the user selects which toolset services they want to use.

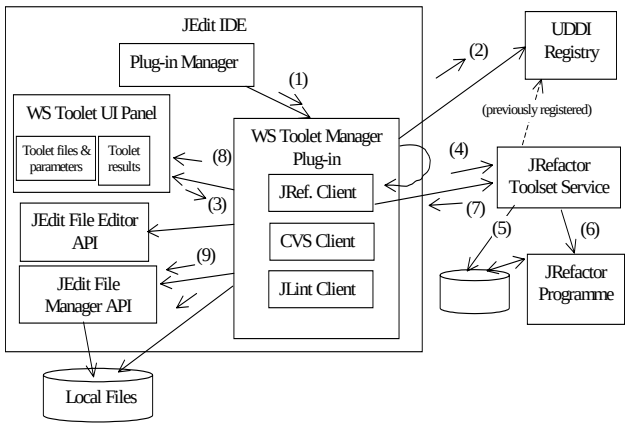

Figure 3. JEdit-WS software architecture.

This results in toolet client adaptors being initialised to communicate with each remote toolset service. A remote toolet service is interacted with by the user via a toolet panel added to JEdit, with request parameters, message text and/or selected files captured to be sent to the remote toolet web service (3). The toolet client adaptor generates SOAP message requests which are sent to the remote toolet service (4). In this example, a request to the JRefactor toolet service includes file(s) to refactor, sent in the SOAP message. These files are stored by the remote service host (5) and then the actual JRefactor programme run over them (6). Results from the remote toolet service are returned to the JEdit web service client (7).

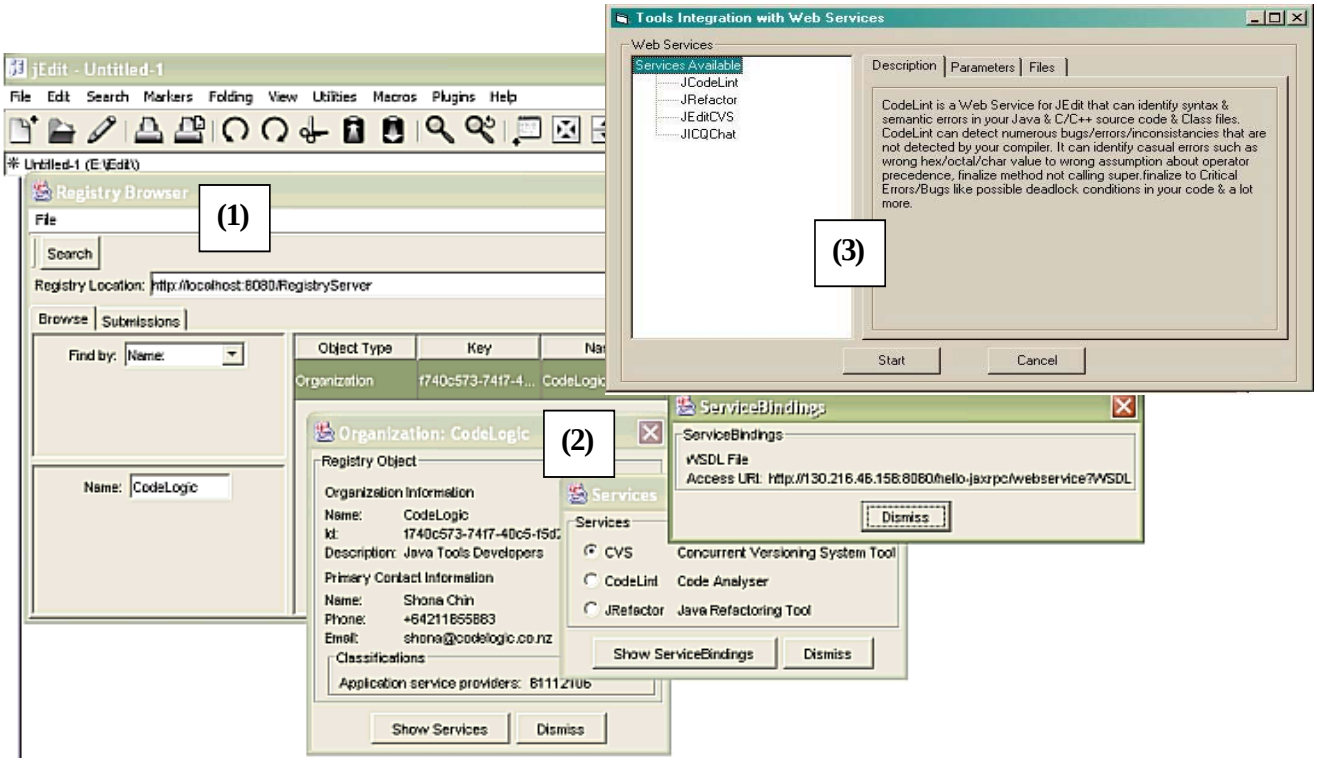

Figure 4. JEdit-WS tool discovery and integration support.

Depending on the kind of toolet service, results may be presented to the user in the toolet panel, in the JEdit editing pane, or the WS toolet manager plug-in may manipulate JEdit data structures e.g. open code files or local PC-held files. In the case of the refactoring tool, a summary of the refactoring is presented (8) and any affected JEdit open files and local PC files updated (9). This approach is suitable for other IDE and CASE tools with plug-in architectures and open APIs, such as Eclipse.

6. Example Usage

In this section we illustrate some example web service-based toolets we have developed for JEdit-WS and how they are integrated and used within the JEdit-WS IDE. Figure 4 (1) shows the interface to the WS toolet plug-in in JEdit-WS. The user specifies a UDDI registry location and this is queried to obtain a list of available toolet services. These can be hosted on the developers own PC or on remote servers. Details of each service can be viewed (2), including the location of the toolet, the particular toolet facilities available and the Web Service Description Language (WSDL) interface description for the services.

After requesting a toolet web service be made available within their JEdit-WS IDE, a user may access the service via a tabbed panel made available by the toolet web service manager plug-in (3). In this example, we have discovered and integrated four toolet services: a code quality assessment tool (JCodeLint), a code refactoring tool (JRefactor), a version control tool (JEditCVS) and an instant messaging tool (JICQChat). Note that there may be more than one available toolet service providing each of these kind of services. In this case the developer would select the particular instance of the remote toolet service that they wish their JEdit-WS IDE to use. Some discovered toolet services may provide a web services protocol that is incompatible with the one encoded in the web service integration plug-in for JEdit-WS. In such a situation the developer may search for a suitable adaptor service to convert the JEdit-WS protocol into the toolet service protocol.

Consider using the developer wanting to make use of the discovered refactoring tool service from their JEdit-WS IDE. This is illustrated in Figure 5. The refactoring tool takes one or more Java source files specified by the JEdit-WS user and runs the JRefactor tool over them, determining a set of appropriate modifications to the source files to achieve various code refactoring improvements. To access this tool the user selects the JRefactor item in the toolet control panel, specifies various configuration parameters for the refactoring process, then specifies file(s) to refactor (1). These files are uploaded to the refactoring tool service where they are processed and potentially modified (2). When finished, a report on the refactorings done and the updated files are returned to the JEdit-WS toolet client (3). Updated files are modified in the JEdit editing buffers and on disk by using the JEdit APIs.

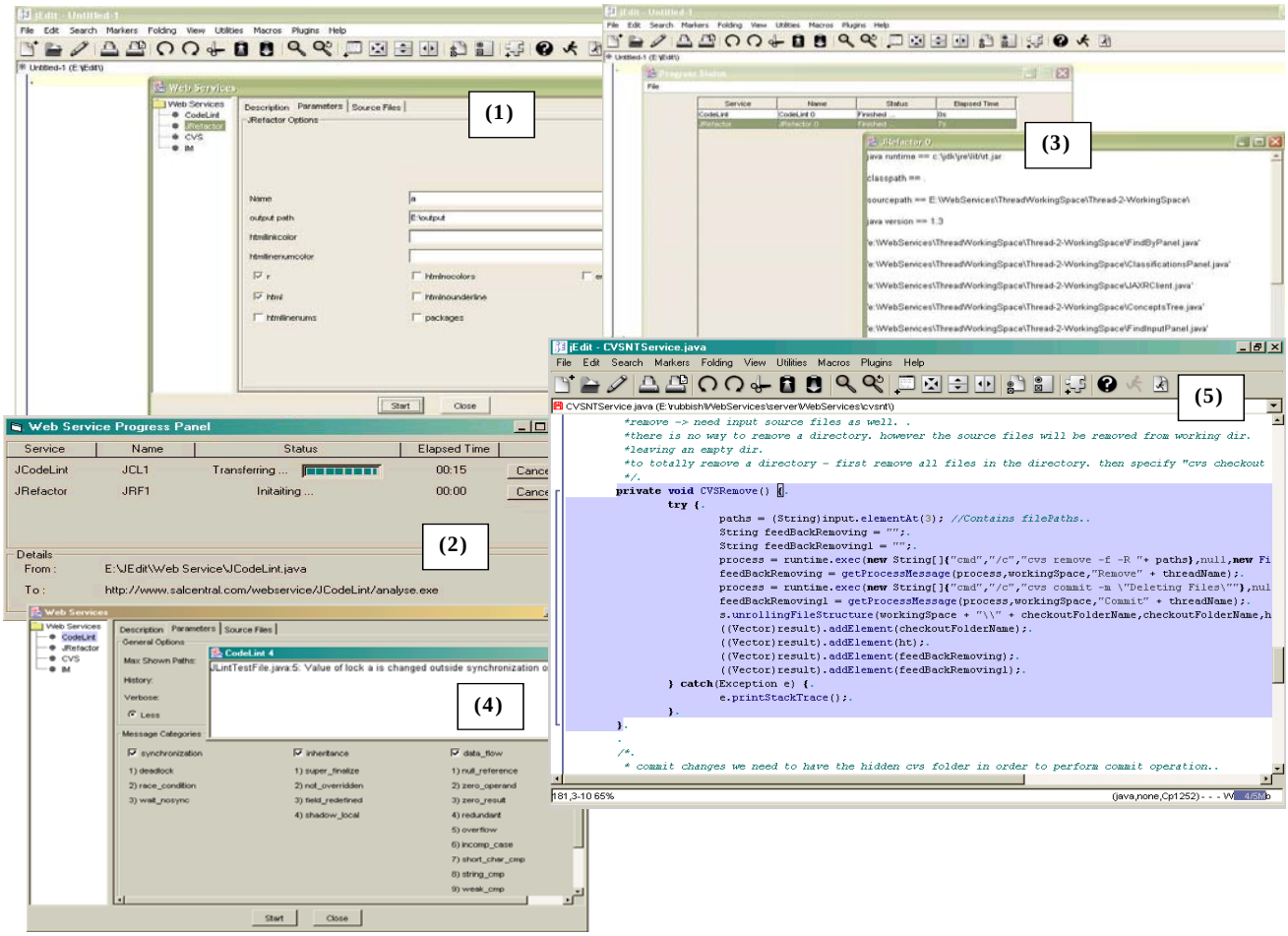

Figure 5. Example of using the code refactoring and code quality assessment toolets.

To support these kinds of interaction with the developer and JEdit, the web service integration plug-in we developed for JEdit-WS provides basic capabilities to interact with the JEdit environment for remote web services. These include asking for toolet service configuration parameters from the user, opening and closing source code files, modifying source code files in the JEdit edit buffers and on the local disk, and displaying results of remote toolet service processing in a window.

Developer interaction with the CodeLint service is similar. The CodeLint toolet is used to assess the quality of Java code and requires one or more Java source code files to process and generates a report, providing line numbers and code quality assessment for the line. To invoke this toolet service the developer selects the CodeLint tool option from the JEdit-WS toolet control panel. They specify one or more Java source code files, either being edited in JEdit or on the local PC, to be assessed. The specified source code file(s) are then uploaded to the remote CodeLint toolet via a SOAP message. The files are assessed by the CodeLint tool and the assessment reports are collected and returned to the JEdit-WS client. The report may be viewed as a text list from the toolet control panel (4), and if the file is open in JEdit the relevant lines of an assessed file are highlighted in the JEdit text editor (5).

The JEditCVS toolet provides version control facilities using a remote toolet web service that itself accesses a shared CVS server. Figure 6 (1) shows the user interface to the version control facility in JEdit-WS. As with the refactoring and code assessment tools, the user can select one or more files being edited within JEdit or stored locally on their PC to check in to the version control tool. They can also browse a list of files held by the CVS version control tool for check out (2). SOAP messages are sent between the JEdit-WS CVS client and the remote CVS toolet service to check in and check out files, as well as to obtain directory listings from the CVS server. Results of version control operations are presented to the user using the toolet control panel (3).

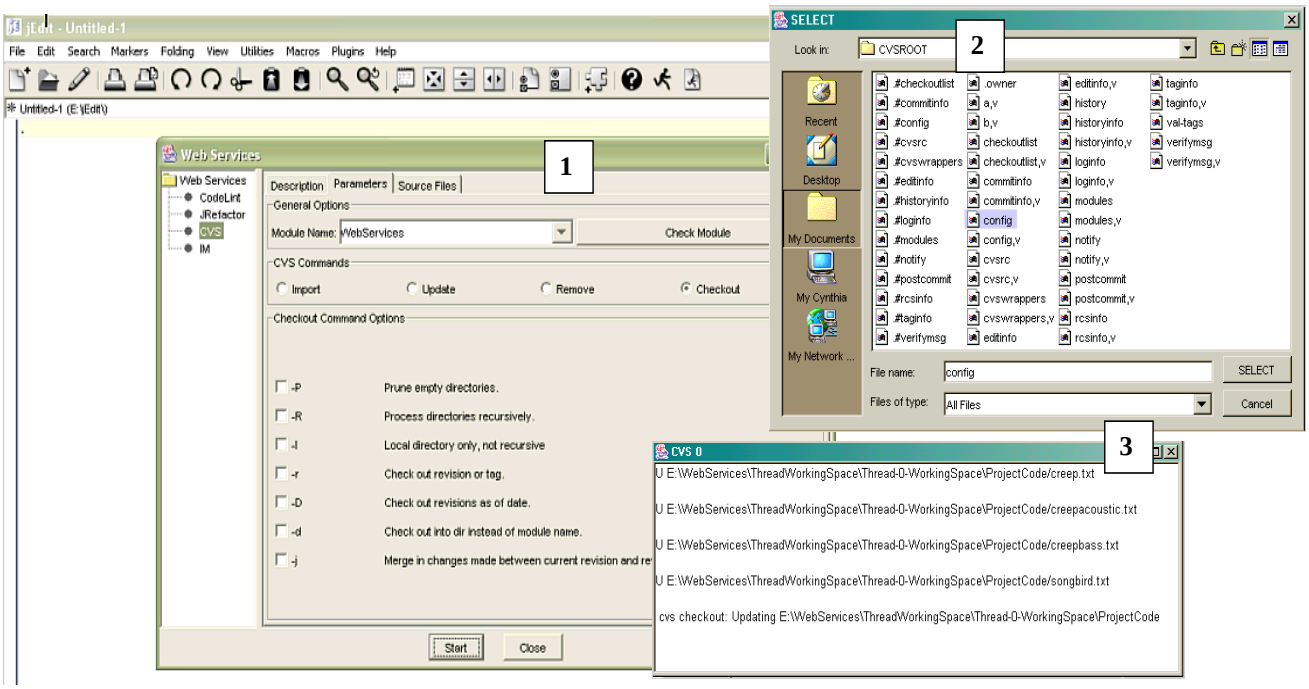

Figure 6. Example of using the version control toolet.

The JRefactor, CodeLint and CVS toolet interfaces in JEdit-WS are all synthesized by the JEdit-WS web service toolet plug-in from the available toolet web service descriptions. Each toolet service has a description providing the user with an overview of the service and how to use it, options for invoking the service (including general toolet options, toolet commands and command parameters), and source and/or result files required by and/or generated by the service. The JEdit-WS toolet service panel generates Description/Parameters/Source Files panel user interface for each service when the toolet service is installed in JEdit-WS. As can be seen from the example panel user interfaces in Figure 5 and Figure 6, these can get quite complex.

The version control tool supports asynchronous collaboration among multiple, distributed developers, who can share files in the CVS repository. In addition, we wanted to provide JEdit-WS users with further support for collaboration, including context-aware messaging and note annotations. We decided to use an existing instant messaging tool server, ICQ, accessed by a remote toolet web service interface. The user logs onto the instant messaging server via the toolet control panel, as shown in Figure 7. They can send and receive messages as in conventional instant messaging tools. They can also have the message history stored by the remote messaging toolet and associated with a file currently open in JEdit-WS, with the message history forming a persistent discussion log associated with this file. This facility can also be used as a "note annotation" facility, where one developer adds a text message against a JEdit file and later on another users sees and reads this message, potentially replying to it.

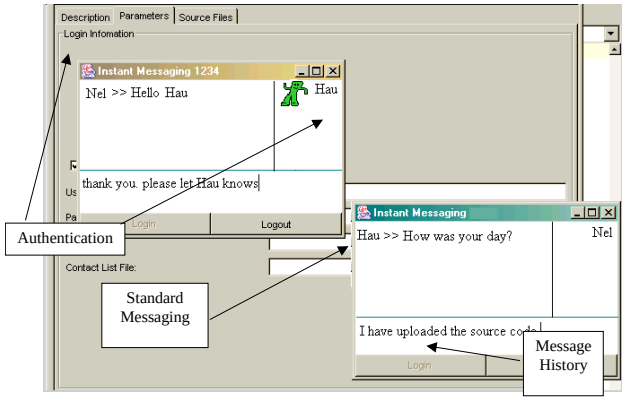

Figure 7. Example of using the text messaging toolet.

7. Design and Implementation

For our proof-of-concept JEdit-WS prototype illustrated in the previous section, we chose to implement a remote web service that takes SOAP messages for multiple toolets and distributes them to each of our web service-enabled toolet services. This is illustrated in Figure 8.

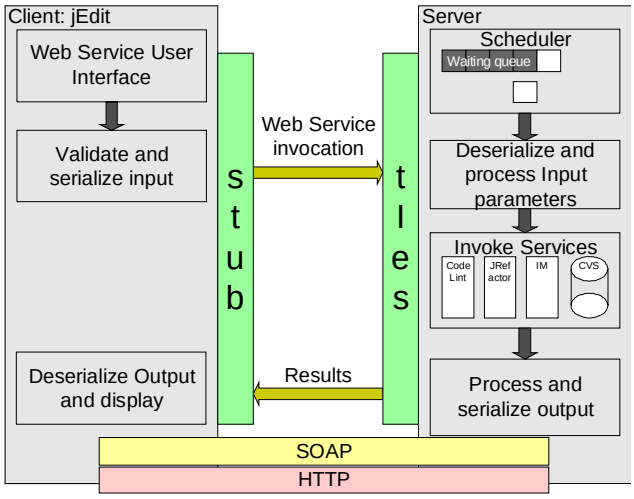

Figure 8. Our prototype JEdit-WS design.

Our JEdit-WS toolet manager plug-in discovers and integrates available services and constructs a basic user interface for each, consisting of a set of tabbed panels including toolet service description , configuration parameters, files to upload/download and results presentation. Our remote service manager behaves as a web service message router, receiving SOAP messages from the JEdit-WS toolet client and scheduling these for processing by our toolet web services. We chose to use this approach to enable large numbers of requests from multiple JEdit-WS users to be processed by multiple service threads. When a toolet web service is invoked by JEdit-WS, a SOAP message including configuration parameters, service request data and possibly one or more Java files is sent to the server, the request scheduled and then data deserialised and the appropriate service invoked. Results are returned via SOAP messages, then deserialised and processed by the JEdit-WS client. Some results are simply presented in a dialogue, some result in JEdit file updates and some in JEdit editor or user interface modifications.

We used a standard release version of JEdit enhanced with a single web service toolet control manager plug-in, using the standard JEdit plug-in API. This locates available toolet web services using a standard UDDI registry and query and allows the user to choose available services for integration. We register all toolet services in the UDDI registry, organising them by simple IDE service categories. WSDL descriptions returned from the registry are used by the JEdit-WS client to assemble a basic adaptor to the remote toolet, including configuring the toolet control panel user interface for the service. Some toolets require update of JEdit IDE information, including highlighting parts of files, updating JEdit file buffers and locally stored files, and displaying chat message dialogue and contents. We used the standard JEdit APIs to achieve these, avoiding code changes to JEdit itself. One challenging implementation issue was encoding Java files in SOAP messages so that invalid control characters didn't affect message deserialisation. If a remote service invocation fails the JEdit-WS client reports this to the user and can either retry requests or attempt to locate and use another instance of the remote toolet service.

8. Evaluation

We carried out three evaluations of our JEdit-WS prototype: a usability survey with several experienced Java IDE users, a performance analysis of the remote services under heavy loading, and a qualitative assessment of the JEdit-WS extensions against alternative approaches to providing these kinds of dynamic IDE extensions.

We surveyed nine experienced Java IDE users, five from the local software industry and four academic staff and post-graduate students. We set development tasks that necessitated the users finding and incorporating these web service IDE extensions. They were asked to use our JEdit-WS IDE to (i) locate a service; (ii) request incorporation of the service; and (iii) make use of the incorporated service in JEdit-WS. They then commented on the suitability of JEdit-WS and its supported web service toolet extensions.

Results of our usability study showed that on the whole experienced Java developers found the approach to dynamically extending JEdit via web service-based toolets to be a viable approach to the problem. The CVS and instant messaging tools were found to be well-integrated and provided an appropriate range of tool facilities and good user interfaces to developers. Some CVS options were found to be difficult to understand and use by developers unfamiliar with CVS and its approach to version control. The CodeLint code quality assessment tool was found to be useful and well-integrated, though most developers surveyed felt the results could be better presented within JEdit code editing windows. Users experience with the refactoring tool was disappointing, with comments including the need for better control over the tool operations and improvements in presenting results to users.

One area of concern with current web service technologies is their likely performance under heavy loading. To assess this for JEdit-WS we developed a performance test suite that made heavy requests to each of the web services, sending a receiving files and reports a large number of times. We measured response time of the services with up to ten concurrent "users" (concurrent threads).

Performance analysis of the toolet web services showed that even our prototype toolet web services could service a moderately large user base. The CodeLint and refactoring tools could easily be run using a large number of multiple threads, while the instant messaging and version control toolets were limited by throughput of their respective third-

party servers. Nevertheless, all of the web service-based tools provided quick (less than 5 second) response to users even when up to ten concurrent requests for each was submitted. It is unclear how the web service-based toolets will perform if large numbers of files have to be exchanged for each interaction e.g. large code base reverse engineering. We think that it is likely they are the wrong solution for such integration problems and a local component is preferable.

Finally, we used a set of qualitative evaluation criteria to assess how well our prototype JEdit-WS IDE satisfied the requirements outlined in Section 2 of this paper. The key criteria we used were:

- Flexibility of the integration mechanism i.e. how wide a range of remote toolet services does it support integration with
- Support for dynamic integration i.e. how easy to find and add new toolet services while the tool is in use
- Sharing of toolet services i.e. how well supported is sharing of a centralised service to reduce need to copy the service and licensing the service
- Development effort for 3rd party integration i.e. how difficult is it to integrate 3rd party, existing toolet services using the JEdit-WS approach
- Usability of the integrated service i.e. how effective and efficient is the generated toolet service user interface in the JEdit-WS IDE

Our JEdit-WS approach allows a wide range of software tool facilities to be dynamically discovered and integrated within an open source, third-party IDE, using the IDE's existing APIs and plug-in management. This was demonstrated by the range of services illustrated in this paper. Remote services can include parameter capture from the JEdit user, JEdit open file and local PC file upload and download, file modification, text message exchange and toolet processing report text. Wrapping existing services was generally straightforward to date, particularly if they provide command-line based control interfaces and read and write files. Our web service toolet control panel provides the user interaction support with remote services It synthesizes a user interface within JEdit-WS to support invocation of each remote toolet service. It makes limited use of the existing JEdit APIs to modify open file edit buffers, locally stored files and to highlight open file contents where these are appropriate for the remote toolet web service. In general services to integrate with our approach need to consume one or more files and produce a report or updated files back to JEdit-WS. However, the ICQ-style chat service shows other kinds of support services can be effectively integrated.

The main weaknesses in our current approach and prototype relate to remote tool parameter setting and understanding by users, and integration of remote tool service input and output into the existing JEdit user interfaces. The version control tool has complex configuration parameters that are hard to understand and use for developers unfamiliar with the tool. The refactoring tool is difficult to both configure and understand its refactoring operations using the current approach of access only via the toolet control panel. The messaging tool user interface could be more tightly integrated with existing JEdit interfaces, such as showing messages inside relevant JEdit code editing windows and buttons to send and view messages via the JEdit toolbar. Our approach is not suitable for all kinds of dynamic IDE and CASE tool extensions. It seems to best suit ones where collaborative information usage is needed, such as versioning, messaging, tool integration, and sharing of information, such as test cases, design patterns and so on. Plug-ins that extend the JEdit environment's local editing and user interface facilities e.g. providing different code viewers, might not suit this approach.

A number of promising current and future research directions exist. The main one is improving the user interface integration of remote toolet services within the JEdit environment. We believe this requires encoding more information about a remote service's user interface needs, both input to and output from the tool, within its WSDL description, and using this to adapt existing JEdit tool interfaces where appropriate. We have used this approach successful in earlier component-based user interface research but not yet with web service-based components [12]. Improved help for users, such as for using configuration parameters and tool output, and semi-automatic use of remote toolet services, also appear to be important future extensions. We need to better support automatic adaptation of JEdit-WS to remote services. For example, if the tool discovers two version control toolet services e.g. CVS and MS Visual SourceSafe™, and these use a different version control service protocol to what the JEdit-WS client adaptor expects. We need JEdit-WS to either discover and integrate an adaptor to translate its protocol to/from the incompatible one, or even to synthesise such an adaptor on the fly. A wide range of other remote toolet services could be developed with this approach. We are currently adding toolet services for a remote JUnit-based test case library manager, basic design pattern management, code generator and CASE tool reverse engineering support. We would like to add further remote code analysis toolets e.g. dependency analysis. We would also like to try and use our toolset services with other extensible Java IDEs, such as Eclipse® and NetBeans®. Once an integrated environment is functioning fully as described above the research progresses to that of demonstrating the real world possibilities in terms of allowing organisations to use a web service driven environment to evaluate new tools, derive useful measurement from usage and provide timely information

regarding usage with regard to remote service (pay per use) vs. permanent licensing.

9. Summary

We have developed a proof-of-concept IDE extension mechanism using web services components. This approach allows an IDE to discover remote "toolet" web services and to integrate and interact with these services from within an open source IDE. We extended the JEdit IDE with a web service toolet plug-in manager, providing toolet service discover, integration, invocation and basic user interface facilities. Some toolets make limited use of JEdit APIs to display results or capture information (Java code) for remote toolet processing. Evaluation of the usability, performance and requirements satisfaction of our prototype toolet services and JEdit-WS IDE have demonstrated the approach is a promising one for remote component-based extension of IDEs and CASE tools.

Acknowledgements

We gratefully acknowledge the support of the Foundation for Research, Science and Technology.

References

1. Bandinelli, S., Di Nitto, E., and Fuggetta, A. Supporting cooperation in the SPADE-1 environment, *IEEE Transactions on Software Engineering*, **22** (12), 1996.
2. Barnes, A. and Gray, J. Workflow products as a tool construction technology for process-centred SEEs, In *Proceedings of 2000 Conference on Software - Methods and Tools*, Australia, Nov 2000, IEEE CS Press.
3. Bitcheva, J., Perrin, O. and Godart, C. Cooperative Process Coordination, In Proceedings of the 1st Int. Conference on Web Services, Las Vegas, USA, June 23-26 2003.
4. Bounab, M., Godart, C. Tool integration in distributed environments: an experience report in a manufacturing framework. *Journal of Systems Integration*, **8** (1), March 1998, Kluwer Academic Publishers, pp.31-51.
5. Dewan, P. and Choudhary, R. Coupling the User-Interfaces of a Multiuser Program, *ACM Transactions on Computer Human Interaction*, **2** (1), 1995, pp. 1-39.
6. Dossick, S.E., Kaiser, G.E. CHIME: A Metadata-Based Distributed Software Development Environment, In Proceedings of the 1999 International Conference on the Foundations of Software Engineering, ACM, pp. 464-475.
7. Ebert, J., Suttenbach, R., and Uhe, I., Meta-CASE in practice: A Case for KOGGE, In *Proceedings of the 9th International Conference on Advanced Information Systems Engineering*, LNCS 1250, Springer-Verlag, Barcelona, Spain, 1997, pp. 203-216.
8. 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*, **3** (3), 213-231, 1997.
9. 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 **42**(2) Jan 2000.
10. Gray, J.P., Liu, A. and Scott, L. Issues in software engineering tool construction, *Information and Software Technology*, **42** (2), Elsevier, 73-77.
11. Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Constructing component-based software engineering environments: issues and experiences, *Information and Software Technology*, **42** (2), January 2000, Elsevier.
12. Grundy, J.C. and Hosking, J.G. Engineering plug-in software components to support collaborative work, *Software – Practice and Experience*, vol. 32, Wiley, pp. 983-1013, 2002.
13. JEdit Homepage, http://www.jedit.org/.
14. Kafeza, E., Chiu, D. and Cheung, S.C. Alert-Driven Process Integration in a Web Services Environment, In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 23-26 2003.
15. Kaiser, G.E. Dossick, S.E., Jiang, W., Yang, J.J., Ye, S.X. WWW-Based Collaboration Environments with Distributed Tool Services, World Wide Web, vol. 1 (1) 1998, pp. 3-25.
16. 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.
17. Kelter, U., Monecke, M. and Platz, D. Constructing Distributed SDEs using an Active Repository, in Proceedings of the 1st International Symposium on Constructing Software Engineering Tools, Los Angeles, 17-18 May 1999, University of South Australia, pp. 149-157.
18. Meyers, S. Difficulties in Integrating Multi-view Editing Environments, IEEE Software, **8** (1), 1991, pp. 49-57.
19. Newcomer, E. Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison-Wesley, June 2002.
20. Pokraev, S., Koolwaaij, J. and Wibbels, M. Extending UDDI with Context-Aware Features Based on Semantic Service Descriptions, Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 23-26 2003.
21. Quatrani, T. *Visual Modeling With Rational Rose™ and Uml*, Addison-Wesley, 1998.
22. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment, IEEE Software, **7** (7), 1990, pp. 57-66.
23. Reiss SP. The Desert environment. *ACM Transactions on Software Engineering & Methodology*, **8** (4), Oct. 1999, pp.297-342.
24. Reps, T. and Teitelbaum, T. Language Processing in Program Editors, *Computer*, **20** (11), 1987, pp. 29-40.
25. Thomas I. PCTE interfaces: supporting tools in software-engineering environments. *IEEE Software*, **6** (6), Nov. 1989, pp.15-23.
26. Thompson AK. CASE data integration: the emerging international standards. *ICL Technical Journal*, **8** (1), May 1992, pp.54-66.
27. Wasserman, A. Tool Integration in Software Engineering Environments, in Software Engineering Environments: International Workshop on Environments, Berlin, 1990, Springer-Verlag.
28. Wilcox et al. A CORBA-Oriented Approach to Heterogeneous Tool Integation; OPHELIA, FSE/SIGSOFT Workshop on Tool-Integration in System Development, Sept 1-2, 2003, Helsinki, Finland.