Reciprocity: an Integrated, Continuous Approach to Software Training Authoring, Delivery and Monitoring

April Nixon[1] and John Grundy[1,2]
Department of Computer Science[1] and Department of Electrical and Computer Engineering[2], University of Auckland, Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

Providing software developers with suitable on-line training support for the tools they use is challenging. We describe Reciprocity, a tool we have developed to support on-line software tool tutorial authoring and usage. Reciprocity supports distributed tutorial construction and viewing, tailored user adaptation, and provides continuous feedback to tutorial authors on the use of their training materials. We illustrate the use of Reciprocity for the construction and use of a tutorial for a domain-specific health data mapping specification tool. We describe our tool's architecture and implementation and report on three evaluations we have carried out to measure its effectiveness.

1. Introduction

Learning to use new computer software is a challenging task for end users but also for software engineers, in terms of the different software development tools they need to learn [18, 12]. Computer-assisted instruction in general can be defined as the use of computers as a medium for delivering instruction. Much research has been carried out in this area, which has seen the development of both research and commercial systems for authoring and delivering computer-assisted instruction, both generic and for specific domains [7, 10, 11]. To date, there has been a limited amount of research into providing software development organizations with tools to support appropriate software training material construction and usage.

Much research has been done in the domain of Computer-Assisted Instruction, where a computer application is taught by means of related computer learning applications [4, 10]. Much of this research has to date been focused on learning environments for algorithms, theories and specific course material [18]. Some limited work has been done on delivering instructional material for computer applications [11, 10]. However, most computer applications continue to be delivered with paper manuals, PDF documents or limited on-line help facilities. These approaches suffer from inconsistent structure and content by multiple authors, lack of feedback to authors from users, difficulty in delivering updated instructional materials to users, and a one-size-fits-all approach to the training material.

In this paper we present a prototype environment for the authoring and delivery of computer-assisted instruction within the domain of software training. Our focus within this area is to test the concept of an integrated authoring and delivery environment focused specifically at software development organisations, which supports continuous service provision and continuous improvement through iteration, real time delivery and reciprocal feedback between learners and authors. To this end we have developed Reciprocity, a software tool tutorial authoring and viewing environment. Reciprocity provides support for distributed software tool tutorial authoring, multi-user tutorial on-line access, and continuous feedback to tutorial authors. The aim of this tool is to enable training materials for software tools to be more easily authored and delivered to users remotely, and user feedback and usage statistics on the training material collected and delivered remotely back to authors. Our goal is to support a continuous cycle of training material delivery, usage, feedback and revision, leading to higher quality training material provision and improved learning outcomes for tool users.

We first provide a motivation for this research from a local Health IT software tool development organization. This company's in-house training group wanted to provide users of their company's software tools an improved on-line tutorial service, and wanted for themselves a better tutorial authoring and usage monitoring system. We review related research in this area and how well these existing tools suit the needs of training material authors. We then show how tutorials are authored in our Reciprocity environment, illustrate how these tutorials are used, and describe how authors gain feedback on tutorial usage. We describe the distributed system architecture and implementation of our tool and report on three evaluations of our tool's effectiveness. We conclude with a summary of the contributions of this research and directions for future research.

2. Motivation

With increased use of computers and software in more occupations and the increasingly business-critical nature of this software, it is important that users receive sufficient training. Orion Systems Ltd (www.orion.co.nz) is a developer of software applications for the health industry. Many of these applications are targeted towards health system developers and integrators i.e. these are domain-specific software tools whose end users and software engineers. One such example is the Symphonia Messaging Toolkit environment, a software tool for the specification of complex data transformations between one health message and another [1]. Orion Systems has an in-house training group responsible for authoring user guides and tutorials for systems such as Symphonia Messaging Toolkit. This team has assembled manual-style documentation and product usage examples, distributed as multiple PDF files with the product.

This approach has a number of limitations, such as large documents to browse, one-size-fits-all tutorials, and the difficulty of updating and extending tutorials and redistributing them to users. The need for a more advanced, learner-centred and timely software training and assistance mechanism was recognised. To this end we explored the development of a task-oriented software training system through a tool to support integrated tutorial building, delivery and monitoring. We then looked at abstracting an improved service model, moving from the current single distribution of a paper-based training manual to a model of continuous service of dynamic training material for Symphonia Messaging Toolkit end users.

Computer assisted instruction, also known as CAI, can be defined as the use of computers as a medium for delivering instruction, and is analogous to computer-based training [4, 16]. Various systems have been developed to support the training of learners of computer software. The SIMPLE environment [10] provides a computer-assisted learning environment by composing multiple existing applications via visual programming techniques. User activity is logged for later playback to instructors to assist in understanding their learning. Tutorials need to be delivered to a student's computer to be used. RIDES [11] provides an environment for building lessons from graphical rules. When building a lesson using RIDES an author can 'record' a procedure that students must learn in a way similar to recording a macro, simply by carrying out the procedure. Students may then carry out the sequence of actions that constitute the task and receive meaningful feedback on the correctness of their actions. Algorithm animation software [2, 8] has been used for many years to help teach learners about computer algorithm behaviour via interaction participation. Interactive illustrations and animations [7, 18] have been used and found to be a very effective means of self-tutoring in a number of computer-assisted learning environments. Web-based software training has become a common approach to providing distributed tutorial material, especially for tertiary distance learning and the corporate training sector [3]. On-line software tutorials have also been tried for software instruction, though seldom for software development tools to date [5, 6]. Most on-line training does not provide feedback to training authors on usage, though many support messaging tools for user questions to experts. Knowledge-based tutors offer another approach to computer-aided instruction [12, 14]. These utilise knowledge of the problem domain, user profile and context in which a software application is to be used to provide more appropriate, context-dependent learning approaches. The main difference between traditional CAI systems and Intelligent Tutoring Systems is the representation of content. In CAI, content is generally designed using a 'storyboard' paradigm, where screens are designed and all possible navigations between these screens are mapped. Although navigation may be non-linear in structure, it is still pre-defined. In ITS's, content is kept separate from the specification of how and when content is presented to the student, allowing multiple use and reuse of content.

In general, most existing CAI approaches focus on localised training programme authoring and subsequent deliver to users' PCs for interaction. There is often a considerable delay in modifying materials and delivering them to users, as with traditional training material delivery via hard copy, Word, PDF and HTML help documents. Web-based training delivery software overcomes this delivery problem but typically does not address the issue of giving feedback to tutorial authors, either implicit usage statistics or explicit tutorial user feedback. Architecturally, a pure web-based delivery system is prone to network failure or slow network performance impacting on the tutorial user's experience i.e. the server hosting the tutorial materials failing or being overloaded by simultaneous requests. Customisation of training materials to a particular user profile may also be limited. A web-based delivery system may not be well-integrated with the target software tool for which it is being used to supply training materials.

3. Outline of Our Approach

Interviews with Orion System's training and support team staff indicated that the current approach they used to supporting Orion's software tool products suffered from several key deficiencies. Authors typically build tutorial material and distribute it as a once-off release with each software product version. Much tutorial material is not integrated with the product as on-line help but rather in stand-along printed manuals of PDF viewer documents. Often authors would structure material differently and inconsistently and much of the material was not focused on learner tasks but rather the software tool features. It was

very difficult for training material authors to evaluate the effectiveness and usability of their tutorials and to respond proactively to different learners' needs.

From these interviews and discussions, we identified the key aims of this work as:

- providing training authors with an environment specifically targeted at producing and maintaining their software tool training materials
- providing a distributed delivery mechanism whereby updates to training material is automatically distributed to all users
- ensuring that users of software training material have up-to-date materials relevant to their own user profiles and software tool training needs
- capturing usage statistics (implicit feedback) and user comments and suggestions (explicit feedback) in a context-aware fashion
- automatically providing this feedback to training material authors so they can use it to proactively enhance their training material

This lead us to the conceptualisation of training material delivery as a "continuous service model" i.e. a cyclical process of authoring->using->feedback->revision. By modelling the provision of software training as a continuous service of dynamic material rather than as a one-off delivery of e.g. a static training manual, we aimed to provide training authors with a much improved environment specifically targeted at producing and maintaining software training material. Key goals of this environment were to:

- reduce the time required to author, update and distribute training material
- make the authoring process more integrated, more user-friendly and the structure and appearance of the material more consistent
- ensure timely delivery of updated training materials to users and feedback from users to authors

The addition of end user tutorial usage and testing reporting, and an open communication channel between authors and learners, aimed to discover if this type of information is useful input into the iterative refinement and evolution of software tool training material.

Figure 1 shows an outline of the process of using our Reciprocity on-line tutorial authoring and viewing environment. A training support person authors tutorials using our tutorial authoring program (1). This includes tutorial structure, text and graphic content, animation, target software instruction, and import of material from other authoring systems. This training material is stored in a authoring database (2), from which it is accessed by tutorial viewing programs. Material can be accessed on-line or the database partially replicated by viewers. Software tool users ("learners") access the tutorials as they require (3), choosing between novice or expert tutorials, tutorials focused on particular tasks, or context-sensitive tutorials as they require. Learners supply information about their training needs and feedback on how helpful they find parts of tutorials, and the viewing program captures usage statistics for each part of a tutorial (4). This learner profile, feedback and usage information is captured in the Reciprocity database (5). As tutorial authors review their material, this information is retrieved for the parts of the tutorial in question (6). The information is presented in-context associated with the tutorial elements in the authoring program (7), and is used to refine the tutorial and respond to the learner feedback.

In the following sections we illustrate the use of Reciprocity for authoring and using parts of our Symphonia Messaging Toolkit tutorials. We then describe key aspects of its architecture and design required to achieve our continuous service model for training material delivery. We report on evaluations of these tutorials by novice and expert users of the software, and then summarise a comparison of its support features against assessment criteria from software instruction literature.

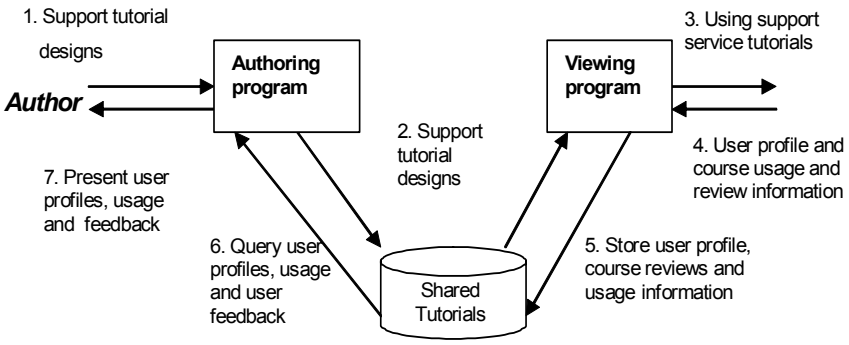

Figure 1. Overview of the Reciprocity on-line tutorial system.

4. Tutorial Authoring

A training tutorial in Reciprocity is organised into Courses, each course having one or more Lessons. Each lesson has a set of subjects making up the lesson that can be viewed in sequence or as required from a lesson index. Each subject has a number of pages and each page contains content such as text, graphics, animations (sequences of pages that can be played), and software tool playback instructions. Each lesson may have specified usage criteria whose statistics the Reciprocity viewer collects as the lesson is used.

Figure 2 shows examples of the authoring tool in use when developing parts of a lesson for the Symphonia Message Toolkit. The authoring tool provides a toolbar for manipulating lesson construction (1) and a tree-based organization of courses and lesson content (2). The content of a selected page is displayed in the page editor (3), in this example containing title, explanatory text and example graphic. Pages have multiple layers of content that may be manipulated to arrange the content (4, 5). Graphics are imported from other tools and a range of formats are supported for playback in the viewer. Examples include bitmap screen dumps, GIF, TIF and JPEG images, PDF, and Flash. This allows authors to use sophisticated content design tools and import their material into Reciprocity tutorials.

One feature of on-line training found to be of particular use for learners in other studies [10, 11] are animations of example scenarios of software applications in use. Reciprocity provides an animation definer (6), allowing the tutorial author to create sequences of animated examples and explanation of software application usage. In this example a step-by-step sequence of the installation wizard for Symphonia is captured, along with example input data from the user, example output from the Wizard and explanation of each step in the animation of using the Wizard. The author captures these example tool screen shots as they use the subject software tool. The learner may step through the animation in sequence, may look at specific steps as they require, or ask Reciprocity to show then the things they need to do when using the target software. The author specifies expected leaner input (mouse click, keystrokes etc) required to advance the animation, which can be used to provide a "simulation" of the target software in use.

Depending on the way the Reciprocity tutorial viewer is integrated with the target software application, such animations may also be used to drive the application itself, having the authoring tool capture interaction events with the target application. When the lesson containing the animation is used from within the target application, the events were then replayed in the target application giving the learner a "live" animation.

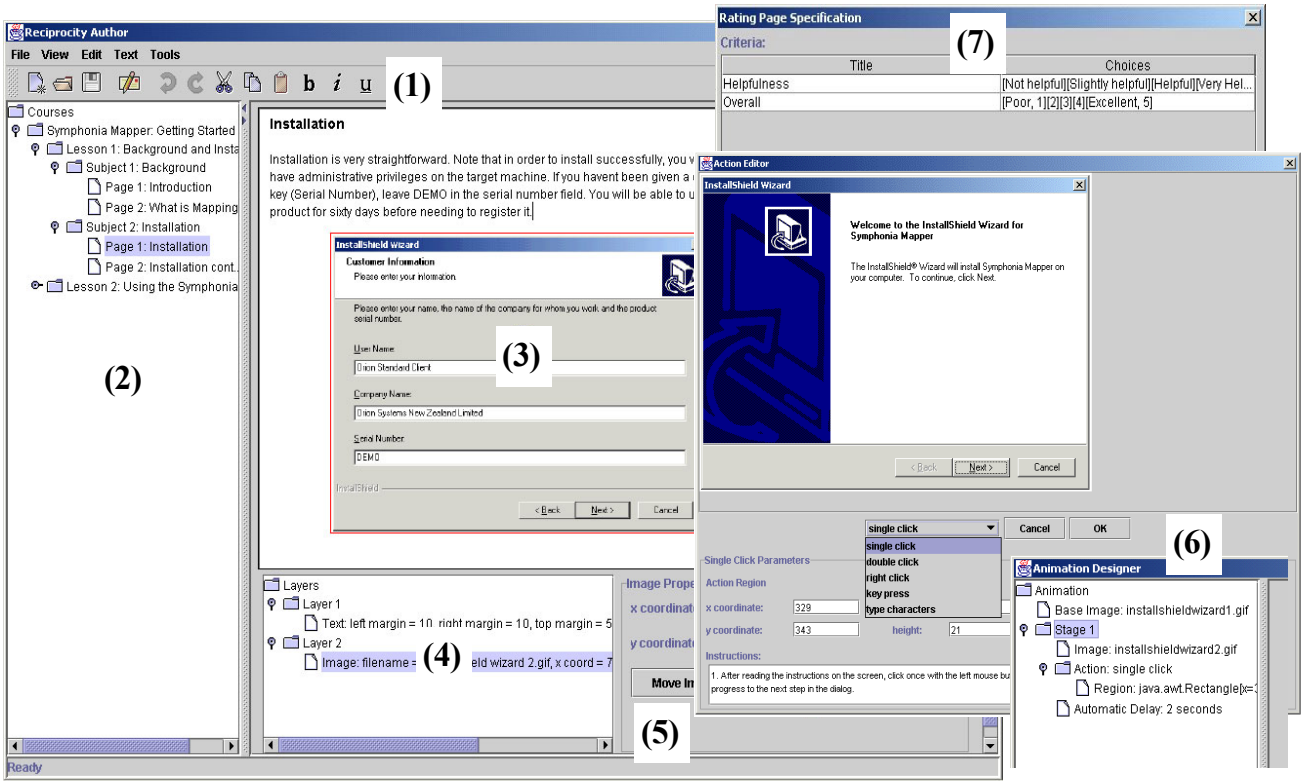

Figure 2. Examples of tutorial authoring in Reciprocity.

Finally, the author can specify a range of monitoring and feedback criteria for parts of a lesson, as shown in (7). This allows the Reciprocity viewer to capture particular requested usage characteristics and the learner can subjectively rate various characteristics. This captured information can then be viewed by the author when reviewing and revising the lessons.

5. Lesson Usage and Monitoring

Packaged courses can be accessed by a viewer organised in two fundamental ways: a separate tutorial viewer application or an integrated component within the target application. We initially built a stand-alone Reciprocity tutorial viewer as a Java application, allowing any software tool tutorial to be authored and viewed. We briefly experimented with a web-based viewer client, to avoid the need for a learner to install a separate viewer application, but this provided much less sophisticated interaction and playback facilities. An integrated viewer, where the viewer forms a component of the target software application itself, is also possible. In this section we describe our stand-alone Java application viewer as it has the most sophisticated functionality.

Figure 3 shows an example of a tutorial from the Symphonia course being used by a software developer wanting to learn to use this tool. When a learner logs into the Reciprocity viewer, they are presented with a list of learner-specific available courses (1). Highlighting indicates which course lessons have been completed, partially completed or not yet used (2). A list of lesson content is shown in a tree structure similar to the authoring program (3), and content in a viewing panel (4). Subjects and pages can be accessed from the tree view or stepped through, as can animations and recorded interaction event sequences (5). The viewer allows the learner to provide closed and open question feedback to the tutorial author on various characteristics (6). An integrated, context-dependent mail tool supports messaging between tutorial authors and learners, automatically capturing the context of the messages (7).

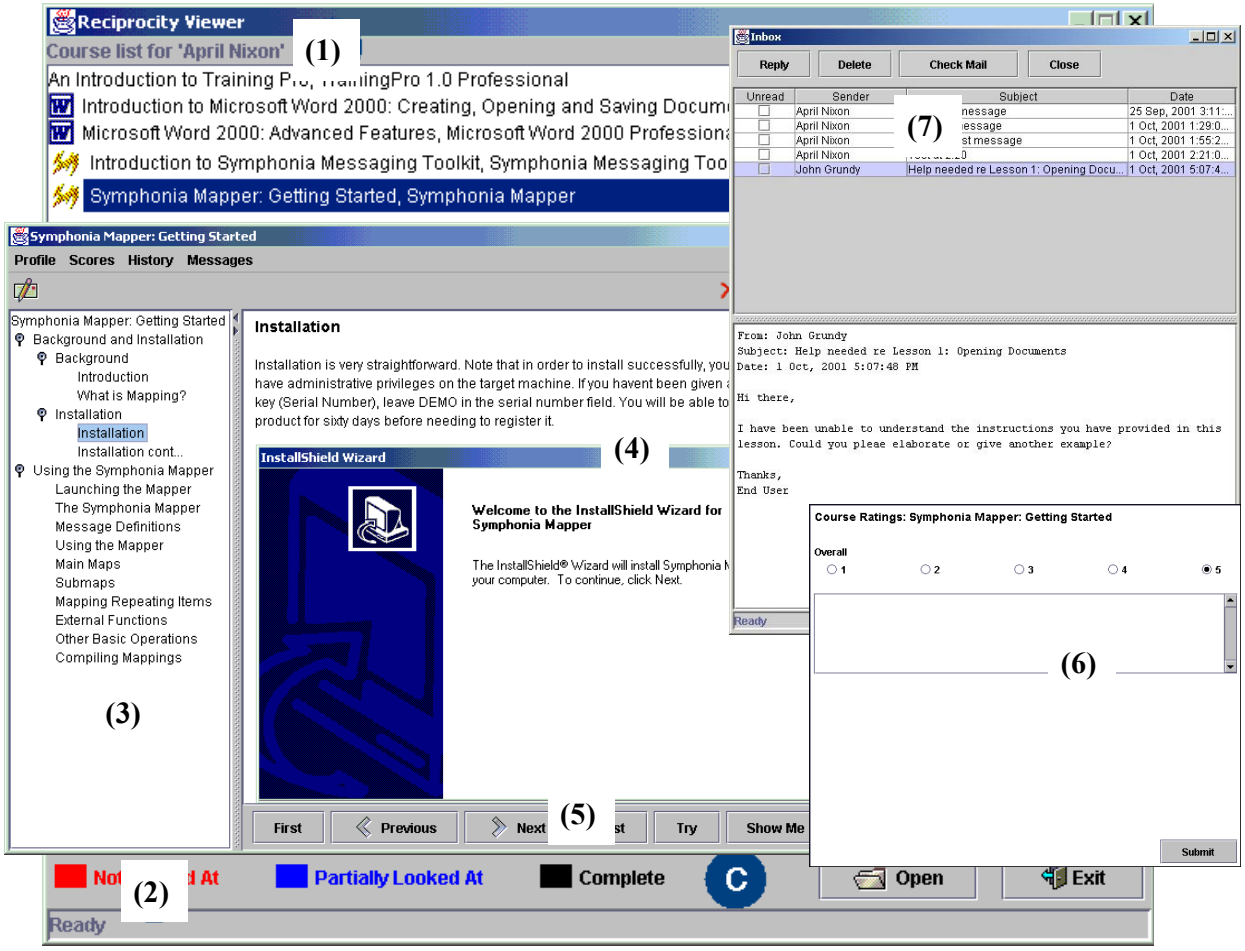

Figure 3. Example of using a Reciprocity tutorial.

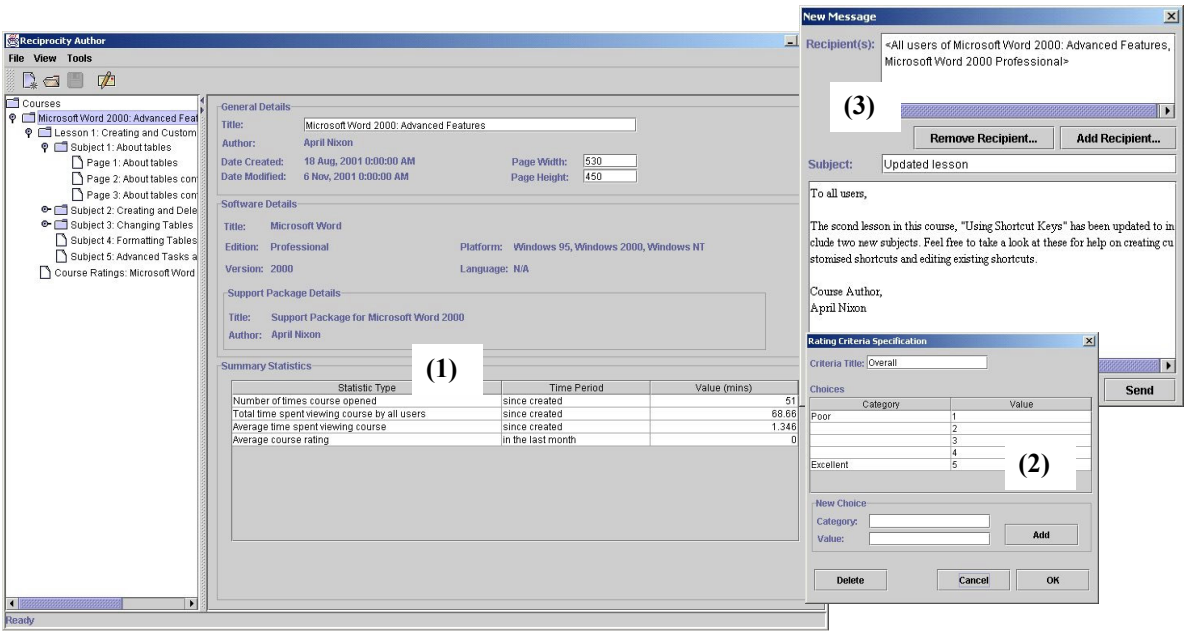

Figure 4. Viewing tutorial usage statistics and feedback.

Learner feedback and usage statistics can be viewed by the author of a course or lesson, as shown in Figure 4 (1). These allow the author to determine when and how tutorial parts have been used, and to interpret learner responses to their courses. The lesson author specifies feedback they would like learners to give them, which might include how helpful a lesson or part of a lesson has been, what information was easy/hard to use, what information is missing and so on. Closed answer responses e.g. "rate on scale of 1-5…" are summarised, while open-answer responses can be browsed anonymously. The author can specify a range of usage statistics for Reciprocity to collect as lessons are used, and these include number of accesses to a course/lesson/subject, distinct learners accessing a course/lesson/subject, and time spent viewing course components. The author may correspond with individual course learners, using a similar integrated, context-aware mail tool as shown in Figure 3, or may broadcast messages to selected learner groups, as shown in Figure 4 (3).

6. Design and Implementation

In this section we describe the architecture and implementation of our prototype Reciprocity on-line software training environment. We needed a distributed environment that allows multiple tutorial authors to build up training materials incrementally, storing them in a shared repository. Users access the training material repository as required. We needed to ensure that down-time of the repository would not unduly impact on users, so local user PC caching of training material would need to be supported when required. In addition, users may want to access the materials via a stand-alone viewer program, a web browser or an integrated viewer built into their target software tool for which they are receiving training. Both implicit and explicit usage information and user feedback are required which must be captured by the user's Reciprocity viewer and fed back to the tutorial author, in the context of the tutorial components being used/commented upon.

Figure 5 shows the main architectural abstractions that make up our Reciprocity prototype. The course authoring client allows the user to create, modify and analyse the content of various courses. It stores all tutorial materials in a shared repository and provides the author with usage statistics and user feednback on tutorial components. It also includes facilities for constructing animations of the target software application being used, which can be replayed by the tutorial users. We also prototyped an extension to this authoring tool that captured basic interaction events from a Java software application which were recorded for replay by learners. This was to demonstrate our architecture could be used to capture and replay actual software tool interaction events if desired.

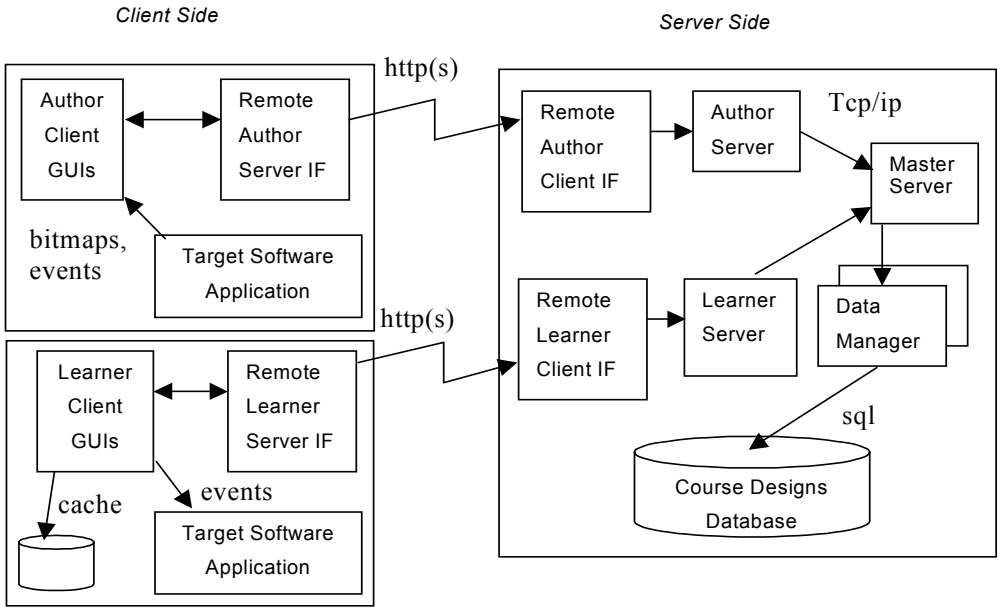

Figure 5. Architecture of Reciprocity.

All course material is stored in a remote repository, accessed by an authoring server. We used a textual http protocol to support author and viewer client interaction with the server, supporting various lesson content distribution and allowing corporate firewalls to be crossed. The author server, which can potentially be replicated and distributed to support large numbers of authors, connects to a master server that provides data storage and retrieval support. A learner server, which again may be replicated and distributed to support a large learner base, provides similar access to the shared course data. The author server can manipulate course data owned by each author, while the learner server can only read data made accessible to each learner and add messages, usage data and learner feedback.

The learner client provides access to the course content, tailored to each learner's needs. A subset of the total number of courses and lessons available is accessible to each learner, and usage history of each is captured for each learner. In addition, each learners feedback on a course's content and their messages related to course content are recorded. An integrated email messaging tool was used to support direct communication with tutorial authors as well as to associate user "comments" with tutorial content.

The learner client may cache parts of the tutorial content to reduce the need to continually download images and animations, and to allow the courses to be used if the learner server should become inaccessible temporarily. This becomes important when a large number of users may otherwise simultaneously access the server, causing response delays. We also developed a prototype event replay facility to demonstrate the target software tool could be driven by the learner client to provide a "live" animation. In this scenario, events captured by the tutorial author using the target application can be replayed by the tutorial user to see their version of the application driven, producing the live animation.

We implemented our Reciprocity prototype using Java applications for the Author and Leaner Clients, Author and Learner servers and Master server. An http protocol is used by the clients to access the Author and Leaner servers, and a custom textual protocol was used to access the Master server functionality. For our prototype we used a Microsoft Access 2000™ relational database to store course content. We used only standard SQL constructs and a more sophisticated database, such as SQL Server™, could be used. We used Swing GUI components to provide the Author and Leaner Client interfaces. A simple web-based viewer was prototyped to experiment with web-based delivery but we found this lacked the interaction support we desired for Learners. The Learner Client was added to a simple Java application to support simple interaction event capture and playback to demonstrate this is feasible with our approach.

7. Evaluation

We have carried out three evaluations of our Reciprocity prototype: two surveys of users of the environment and one qualitative evaluation using two heuristic approaches, one focusing on the integration of learning and usability when evaluating educational multimedia software [Squires 99], and the other on general software usability [Nielsen 94]. Our aim was to gauge the degree of assistance to novices in the first evaluation and to get expert feedback from experienced training material

authors at Orion Systems in the second survey. The qualitative evaluation assessed Reciprocities usability against general criteria and its support for learning against instructional software assessment criteria.

For the first survey, we used a group of novice Symphonia users, staff and graduate students from the University of Auckland. The aim of this survey was to simply gauge the suitability of the environment's facilities for general software training support. For the second survey, we used staff from Orion Systems and a comprehensive set of Symphonia lessons. Some Orion staff were experienced users of the tool, including their training support team, typical of our intended course authors. Others had not used it and were representative of the target software developers (i.e. our target tutorial users) for who the message mapping toolkit was designed. The aim of this second survey was to gauge the suitability of the tool for novice to expert users in the target domain, both trainers ("authors") and programmers ("users"). Both Authoring and Learner clients were used in both surveys and users carried out lesson authoring and learning tasks then completed a post-task questionnaire.

In the first survey we had the user group rate the usefulness and importance as well as experienced usability of the Author and Learner client programs. On the whole, all features of the current prototype tools rated very well. A number of key authoring tool features that were desired or that users felt required enhancement were identified. These included support for more flexible overviews of content and storyboards, hyperlinks between course elements and improved graphical reporting of course rating and usage statistics. Viewer features that were identified as needing improvement included supporting wider range of content, target application integration and addition of ability to sit a test of learned material. In our second survey, the current prototype course structuring and page layout specification features were rated rather lower and experienced training material authors used to high quality authoring tools suggested these required some improvement. Hyperlinks, improved media support (sound, video) and the ability to create tests for Learners were all also rated as highly desired features to add to the tool. For the Learner client, improvements to navigation support and media support were the key features desired by the Symphonia Messaging Toolkit programmers using Reciprocity to learn how to use this tool.

We carried out a further evaluation of our Reciprocity prototype by employing a qualitative evaluation of the environment using two sets of evaluation criteria. The first a set of heuristics we used [Squires 99] focus on the integration of learning and usability when evaluating educational multimedia software. These criteria include assessing the appropriateness of the complexity of the multimedia used, learner activity when using the tutorial, use of fantasy in the tutorial metaphors, software navigatability, and learner feedback, motivation and control. The second set of heuristics [Nielsen 94] provide a more general evaluation the usability of software applications. These include measures such as visibility of system status, match between the system and the real world, user control and freedom, consistency and use of standards, error prevention and recovery support, and flexibility and efficiency of use.

Our qualitative evaluation found that, using Squires' learning assessment criteria, the environment supports integrated learning and usability well. Key advantages include encouraging active users, appropriate use of multi-media, simple navigation and supporting learner feedback to authors. Areas for improvement include more flexible navigation support, allowing learners to "try again" during animated demonstrations if they make a mistake, and provision of a testing facility for learners to gauge their progress. These results concurred with the feedback from the survey of Orion's training team members and further discussions with team members. From our general usability criteria assessment, the tool provides a good match between real world and system concepts (by using terminology from paper-based manuals), good user control, enforces consistency of structure of tutorials, and uses simple and clear designs for authoring and viewing interfaces. Areas for improvement include error prevention, undo/redo of actions, and improved flexibility of navigation between parts of tutorials. Integration of the viewer and target software tool may also improve aspects of tutorial usage for the user.

We are planning to investigate ways to integrate Reciprocity with other applications to support better capture and reply of actions. We plan to use web service-based technologies to support remote application interaction to achieve this, and are planning to integrate the viewer with a meta-CASE tool we are developing using this approach. The addition of a self-test facility [9] to enable learners to gauge their progress and potentially to allow authors to view summarised test results for a group of learners is a facility our evaluations suggest could greatly enhance Reciprocity's support for learning.

8. Summary

We have described a new on-line training material authoring and viewing environment, Reciprocity. This tool supports the creation and evolution of training materials for software applications and multi-user distributed access to this material. Learner profiles allow tailoring of tutorial materials to different groups and individuals, and feedback to authors from learners includes both explicit ratings, messaging and usage statistics. Evaluation of a prototype of Reciprocity used to support a Health IT software provider's message toolkit application tutorials has demonstrated the tool provides useful support for continuous software training material update, distribution and usage.

Acknowledgements

The assistance of Orion Systems Ltd in supporting this research is gratefully acknowledged, particularly the involvement of Dave Brewerton in acting as mentor for the project. April Nixon was supported by a Technology in Industry Fellowship from Technology New Zealand.

References

1. Brewerton, D. Symphonia 3 Training Manual, Orion Systems New Zealand Limited, www.orion.co.nz, 2001.
2. Byrne, M., Catrambone, R. and Stasko, J.., Evaluating Animations as Student Aids in Learning Computer Algorithms, Computers & Education, Vol. 33, No. 4, 1999, pp. 253-278.
3. Coppola, N. W., Myre, R. Corporate software training: is Web-based training as effective as instructor-led training? IEEE Transactions on Professional Communication, vol. 45, no.3, Sept. 2002, pp.170-86.
4. Demetry, J.S., Black, B., Voltmer, D., Nahvi, M., Jones, J. Computer-Assisted Interactive Instruction: Results from a Developmental Effort, Frontiers in Education, 1992. Proceedings. Twenty-Second Annual Conference, Pages 662 –667.
5. Foster, G. Online help systems: learning while working, In Proceedings of the 2002 International Conference on Computers in Education, Auckland, New Zealand, 3-6 December 2002, pp.652-653.
6. Garcia-Crespo, A., Domingo, P., Lancha, M., and Ruiz-Mezcua, B. EDU-EX: an intelligent educational system generator tool over the Web, In Proceedings of the 1997 Conference on Intelligent Information Systems, IEEE CS Press, pp.503-506.
7. Gould, D.L., Simpson, Rosemary M., and van Dam, A. Granularity in the Design of Interactive Illustrations, Proceedings of ACM SIGCSE 1999.
8. Hundhausen, C., Douglas, S., and Stasko, J., A Meta-Study of Algorithm Visualization Effectiveness, Journal of Visual Languages and Computing, vol. 13, no. 3, June 2002, pp. 259-290.
9. Hussmann, S., Covic, G. and Patel, N. Effective Teaching and Learning in Engineering Education using a novel Web-based Tutorial and Assessment Tool for Advanced Electronics, International Journal of Engineering Education, 2002.
10. Marcy, W.M., Hagler, M.O. Implementation issues in SIMPLE learning environments, IEEE Transactions on Education, Volume: 39 Issue: 3, Aug. 1996 Page(s): 423 –429
11. Munroe et al, A Tool for Building Simulation-Based Learning Environments, Behavioural Technology Laboratories, University of Southern California.
12. Murray, T. Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design, Journal of the Learning Sciences, 1998. Vol. 7, No.1, pp. 5-64.
13. Nielsen, J. Usability inspection methods, in: J. Nielsen, R.L. Mack (Eds), Usability Inspection Methods, John Wiley, New York, 1994, p. 30.
14. Petropoulakis, L., McArthur, S., McDonald, J., Agent-controlled internet tools for computer-based distance training in industry and education. International Journal of Continuing Engineering Education, vol.12, no.1-4, 2002, pp.267-276.
15. Reeves, T.C., Evaluating interactive multimedia, Educational Technology, May p. 47-52.
16. Spalter, A.M, Simpson, R.M., Legrand, M., Taichi, S. Considering a full range of teaching techniques for use in interactive educational software: a practical guide and brainstorming session. 30th Annual Frontiers in Education Conference, IEEE CS Press, Vol.2, 2000, Champaign, IL, USA, pp. 19-24.
17. Squires, D. and Preece, J. Predicting quality in educational software: Evaluating for learning, usability and the synergy between them, Interacting with Computers 11 (1999) p. 467-483.
18. van Dam, A. Education: the Unfinished Revolution, ACM Computing Surveys (CSUR) December 1999.