HAL
open science

Wooki: A P2P Wiki-Based Collaborative Writing Tool

Stéphane Weiss, Pascal Urso, Pascal Molli

To cite this version:

Stéphane Weiss, Pascal Urso, Pascal Molli. Wooki: A P2P Wiki-Based Collaborative Writing Tool. 8th International Conference on Web Information Systems Engineering – WISE 2007, Dec 2007, Nancy, France. pp.503-512, 10.1007/978-3-540-76993-4_42 . inria-00432360

HAL Id: inria-00432360

https://inria.hal.science/inria-00432360

Submitted on 16 Nov 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Wooki: a P2P Wiki-based Collaborative Writing Tool

Stéphane Weiss, Pascal Urso, and Pascal Molli

Nancy-Université, LORIA, INRIA-Lorraine
{weiss,urso,molli}@loria.fr

Abstract. Wiki systems are becoming an important part of the information system of many organisations and communities. This introduce the issue of the data availability in case of failure, heavy load or off-line access. We propose to replicate wiki pages across a P2P network of wiki engines. We address the problem of consistency of replicated wiki pages in the context of a P2P wiki system. In this paper, we present the architecture and the underlying algorithms of the wooki system. Compared to traditional wikis, Wooki is P2P wiki which scales, delivers better performances and allows off-line access.

1 Introduction

Currently, wikis are the most popular collaborative editing systems. They allow people to easily create and modify content on the web. This ease of interaction and operation makes a wiki an effective tool for collaborative writing. Collaborative writing is becoming increasingly common; often compulsory in academic and corporate work. Writing scientific articles, technical manuals and planning presentations are a few examples of common collaborative writing activities.

A lot of critical data are now under the control of wiki systems. Wikis are now used within enterprises or organizations. For example, United States intelligence community uses Intellipedia for managing national security informations. Wikis are now an important piece in the information system of many large organizations and communities. This introduces the issue of data availability in case of failure, heavy load or off-line access.

Current wiki systems are intrinsically centralized. Consequently, in case of failure or off-line work, data are unavailable. In case of heavy load, the system scales poorly and the cost linked to underlying hardware cannot be shared. Our objective is to replace the centralized architecture of a wiki server by a P2P network of wiki servers. This makes the whole wiki system fault tolerant, this allows to balance the load on the network and finally costs of the underlying hardware can be shared between different organizations.

This approach supposes that wiki data are replicated on a P2P network of wiki sites. Consequently, the main problem is how to manage replicate consistency between wiki sites. Traditional pessimistic replication approaches (Distributed Database, ...) ensure consistency but are not adapted to this context.

They scale poorly and do not support off-line work. Optimistic replication approaches suppose to know how to safely merge concurrent updates. Some previous work tried to build a P2P wiki [1, 2] relying on distributed version control system (DVCS) approaches. The main problem with DVCS approach is correctness. An optimistic replicated system is considered as correct if it eventually converges i.e., when the system is idle all sites contain identical data. This is called *eventual consistency* [3]. DVCS have never ensured this property. Consequently, building a P2P system with a DVCS will not ensure eventual consistency. Other approaches ensure convergence [4, 5] but are not compatible with P2P networks constraints. Finally, other approaches converge and are adequate with P2P constraints but do not support collaborative editing constraints.

We developed the woot [6] algorithm to manage consistency of a replicated linear structure in a P2P environment. This algorithm ensures convergence without managing versions. In this paper, we describe how we built Wooki. Wooki is a fully functional P2P wiki system based on this algorithm. We refined the original algorithm to achieve a better time complexity. We combined this new algorithm with a probabilistic dissemination algorithm for managing updates propagation on the overlay network and with an anti-entropy algorithm for managing failures and disconnected sites. Wooki is currently available under GPL license.

2 The Wooki approach

A wooki network is a dynamic p2p network where any site can join or leave at any time. Each site has a unique identifier named *siteid*. Site identifiers are totally ordered. Each site replicates wiki pages of other sites. Each site only requires a partial knowledge of the whole network.

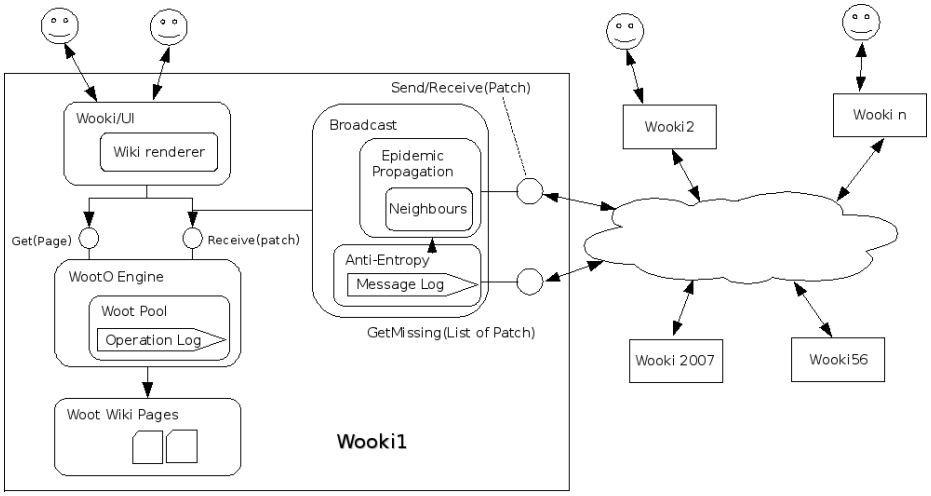

Fig. 1. Wooki architecture

There is three main components in a wooki site (figure 1). The core component wooto which is in charge of generating and integrating operations affecting the documents. Another component is in charge of user interface. The last component is in charge of disseminating local operations and retrieving remote operations.

2.1 Wooto Approach

Wooto is an optimized version of woot [6]. A wooki page is identified by a unique identifier *pageid*. This identifier is set when the page is created. This identifier is the name of the created wiki page. If some sites create concurrently pages with the same name, their content will be directly merged by the wooto algorithm. A wooki page contains a sequence of four-tuples $< idl, content, degree, visibility >$, where each tuple represents a line of the wooki page.

idl is the unique identifier of the line. This id is a pair $(siteid, logicalclock)$. Each site maintains a logical clock [7]. Each time an operation is generated, the logical clock is incremented. The line identifiers are strictly totally ordered. *Let* $idl1$ *and* $idl2$ *be two line identifiers with their respective values* $(s1, l1)$ *and* $(s1, l2)$. *We get* $idl1 <_{id} idl2$ *if and only if (1)* $s1 < s2$ *or (2)* $s1 = s2$ *and* $l1 < l2$.

content is a string representing the content of the wiki-readable line.

degree is an integer used by the wooto algorithm. The degree of a line is fixed when the line is generated. We will describe how the degree is computed when we will describe the generation of an operation.

visibility is a boolean representing if a line is visible or not. It means that in the wooki approach, we do not delete lines, we mark them as invisible.

By applying this storage model, if we have the following wiki page:

```
=== Three pigs ===
* [[Image:pig.png|thumb|left|100px|riri]]
fifi and loulou
```

Assuming these three lines were generated on site number 1 in this order and assuming there is no invisible lines, the wiki page will be internally stored as:

```
((1,1),=== Three pigs ===,0,true)
((1,2),* [[Image:pig.png|thumb|left|100px|riri]],1,true)
((1,3),fifi and loulou,2,true)
```

To manage the storage model, the Wooki handles two operations: the insertion and the deletion of a line. As in traditional version control systems, there is no operation of line update.

1. $Insert(pageid, line, l_P, l_N)$ where *pageid* is the page where to insert the line $line =< idl, content, degree, visibility >$. l_P and l_N are the id of the previous and the next lines.

2. $Delete(pageid, idl)$ sets visibility of the line identified by idl as false in the page identified by $pageid$. Optionally, the content of the deleted line can be garbaged. However, to maintain consistency, the deleted line identifier must be kept as *tombstones*. Tombstones, also known as "death certificate", are heavily used in optimistic replication, especially in Usenet [8] or replicated databases [9].

When wooto creates a new page, the page is initialized as : L_B, L_E. L_B and L_E are special lines indicating the beginning and the ending of a page. When site x generates an insert operation on page p, between $lineA$ and $lineB$, it generates $Insert(p, < (x, ++clock_x), content, d, true >, idl(lineA), idl(lineB))$ where $d = max(degree(lineA), degree(lineB)) + 1$. The lines L_B and L_E have a degree of 0. The degree represents a kind of loose hierarchical relation between lines. Lines with a lower degree are more likely generated earlier than lines with a higher degree.

Every generated insert operation must be integrated on every wooki site (including the generation one). The wooto algorithm is able to integrate insert operations and to compute the same result whatever the integration order of insert operations. In the following algorithm, S represents, on the local site, the lines sequence of the page where a line have to be inserted.

```
IntegrateIns(l, l_p, l_n) :-
  let S' := subseq(S, l_p, l_n);
  if S' := ∅ then
    insert(S, l, position(l_n));
  else
    let i := 0, d_min := min(degree, S');
    let F := filter(S', λl_i. degree(l_i) = d_min);
    while (i < |F| − 1) and (F|i| <_id l) do i := i + 1;
    IntegrateIns(l, F[i − 1], F[i]);
  endif;
```

This algorithm selects the sub-sequence of lines present between the previous line and the next line. If this sequence is empty, the line l is inserted in the model just before the next line. Elsewhere, wooto filters the sub-sequence keeping only lines with the minimum degree. The remaining lines are sorted according the $<_{id}$ order [6]. Thus, l must be integrated at its place according $<_{id}$ between remaining lines. We then make a recursive call to place l among lines with higher degree.

However, since wooki sites can receive operations in any order, the operations have pre-conditions. When a site receives an operation, if its pre-conditions are verified, the operation can be integrated immediately by executing the wooto algorithm. If pre-conditions are false, the operation is placed on a waiting queue until its pre-conditions are verified. Pre-conditions are: a line can only be inserted on a site if its previous and next lines are already present in the model of this site. Similarly, only an existing line can be deleted.

Let's now illustrate wooto algorithm through an example (see figure 2). On this scenario, line L_1 and line L_2 were concurrently inserted when the page was empty, and line L_3 was inserted before line L_1. Without woot or

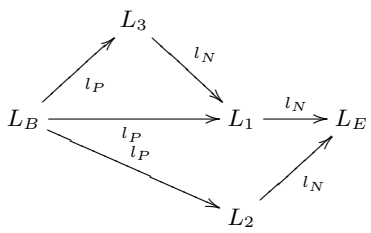

Fig. 2. Example

wooto, such a scenario can lead to three different line orders L_B, L_2, L_3, L_1, L_E, L_B, L_3, L_2, L_1, L_E or L_B, L_3, L_1, L_2, L_E each of them respecting the previous and next relationship. Wooto computes the unique order L_B, L_3, L_1, L_2, L_E.

According to definition, we have $degree(L_1) = degree(L_2) = 1$ and $degree(L_3) = 2$. Due to pre-conditions, there is no site where line L_3 is present and not line L_1. Now, let's assume that $L_1 <_{id} L_2 <_{id} L_3$.

Imagine a site, where line L_2 arrives when line L_1 and line L_3 are present. The wooto algorithm first selects the lines present between L_B and L_E and filters them to keep only line L_1. The reason of such a filtering is above. Indeed, since line L_3 is dependent to another line – here line L_1 –, it has a higher degree. Moreover, there could exist a site where line L_1 is present and not line L_3. Thus, line L_2 must be placed according to line L_1 earlier to line L_3. We obtain L_B, L_3, L_1, L_2, L_E. The wooto algorithm computes the same order L_B, L_3, L_1, L_2, L_E whichever the integration order respecting pre-conditions.

Finally, compared to woot original integration algorithm, the filtering is done on degree instead of the previous and next relationship of each line. Thus, wooto has a better time complexity : $O(n^2)$ instead of $O(n^3)$. This also allows to reduce the space required to store lines: an integer replacing two line identifiers. As woot, we have formally verified the eventual consistency of wooto with the Lamport's model-checker TLC [10].

2.2 User's operation

Users do not directly edit the model. When a user opens a page for editing, he sees a view of the model which presents only the content of the visible lines. As in traditional wiki, the user makes all the modifications he wants and saves.

To detect operations, we use a diff algorithm [11] between the page the user requested at edition time and the page the user saves. We translate the operations given by the diff algorithm in terms of wooto operations. A delete of the line number n is translated into a delete of the nth visible line. An insert at position n is translated into an insert between the $(n-1)^{th}$ and the nth visible lines. These operations are integrated locally and broadcasted to the other wooki sites.

2.3 Wooki Broadcast

For operation dissemination, we use a broadcast protocol in the spirit of [12]. We combine a probabilistic broadcast algorithm with an anti-entropy algorithm [9]. The probabilistic broadcast quickly disseminates updates on the P2P network and to manage membership. The anti-entropy algorithm has the responsibility to recover missing updates for sites that were off-line or crashed.

In order to be deployed on a P2P network, the broadcast protocol must be scalable, reliable and must also support the *churn* of the network. Indeed, in a P2P network, nodes join and leave the network dynamically. Thus, we replace the standard probabilistic broadcast of [12] by the lightweight probabilistic broadcast (lpbcast) [13]. This algorithm gives probabilist warranties that sent operations will be delivered to all connected nodes.

In order to ensure a reliable dissemination of messages, each site must manage a table of neighbors. This table has a fixed size and contains only a partial nodes list of the entire P2P network. Lpbcast updates this table during messages propagation. Lpbcast gives probabilistic warranties that there is no clusters within P2P network. Moreover, since wooto does not require an ordering on message reception, the lpbcast can be unordered for higher efficiency.

The lpbcast algorithm ensures a reliable and scalable dissemination of operations on connected nodes. For managing off-line work, we combine it with an anti-entropy algorithm. We use the original anti-entropy algorithm of [9]. When a site starts an anti-entropy, it selects a neighbor at random in the local table of neighbors and send to him a digest of his own received messages. The selected site returns missing messages to caller. Using anti-entropy implies that each site keeps received messages in a log. As this log should not grow infinitely, we purge this log from time to time. Purging the log is an intrinsic problem of anti-entropy approach. If a site purges its log and then starts an anti-entropy with another site, it can receive previously purged messages. In the traditional approach, tombstones or death certificates are used to avoid this problem. Fortunately, Wooki supports re-integration of already integrated operations. Indeed, we drop an insert operation of a line which identifier is already present in the model. Also, reapplying deletion of a line causes no modification. Thus, we can purge the anti-entropy log without using operation tombstones. This is an interesting property of combining wooto and anti-entropy approach.

Finally, if a site is off-line for a long period of time, anti-entropy may not find missing messages i.e., missing messages which have been purged on all sites of the P2P network. In this case, this site cannot be synchronized. The only way to recover this site is to make a state transfer with a site. It is important to notice that, even in this case, off-line work is not lost since local operations are still applicable. The site can send its local operations and then make the state transfer.

Fig. 3. Wooki Interface

Fig. 4. Wooki Neighbors

3 Implementation

The wooki prototype has been implemented in Java as servlets in a Tomcat Server. Wooki pages are just stored in regular files. All network messages are transported by the http protocol.

In figure 3, we can see the same page entitled "wooki1" loaded from 2 different wooki sites: http://wooki.loria.fr/wooki1 and http://wooki.loria.fr/wooki2. In this example, the same server "wooki.loria.fr" hosts 3 wooki sites. We can also see on figure 3 that both sites are connected and each site has two neighbors. We can observe the table of neighbors in figure 4. This interface lets the administrator manage manually this table. In the normal case, no administration is required except when starting wooki for the first time. The administrator has to connect the new wooki site to an existing wooki network by typing the http address of a wooki node. Once bootstrapped, the routing table is updated when messages from other sites are received. This interface also allows the administrator to start an anti-entropy mechanism or a state transfer. In the normal case, anti-entropy is activated at a regular interval of time. Administrator can force anti-entropy when he restarts the wooki system or when an off-line session is ended.

The wooki prototype is available with a GPL license at http://p2pwiki.loria.fr.

4 Related work

Pessimistic replication (database [14] or consensus [15]) ensures that, at anytime, all replicas host the same content. Pessimistic replication is widely used and recognized for its safety. Unfortunately, pessimistic replication is also well-known for its poor scalability [3] and its incapacity to provide an off-line work.

Virtual Synchrony [16] is a middleware which allows sharing data among programs running on multiple machines. Virtual Synchrony provides a pessimistic replication with some optimizations to improve global performance. Consistency is ensured by enforcing a total ordered reception of modifications. Unfortunately, this ordering is costly and do not scale to a huge number of sites. In addition, Virtual Synchrony does not allow disconnected work.

Icecube [17] is an optimistic reconciliation algorithm. Using static or dynamic reconciliation constraints, some semantic relations can be defined between users' operations. The Icecube algorithm aimed to obtain the best operations' schedule satisfying all constraints. Therefore, Icecube requires a central site which have to choose this schedule. Unfortunately, a central site is a serious bottleneck, hence, Icecube is not well-fit to provide a scalable wiki. In addition, due to concurrency, some constraints may not be satisfiable, hence, some operations have to be dropped. On the contrary, in the wooto approach, operations never conflict and, consequently, they are all integrated.

Joyce [5] is a programing framework based on the Icecube approach. In Joyce, the schedule is incrementally determined. Using Joyce, the authors proposed a collaborative text editor called Babbles which supports selective undo/redo mechanism. Thanks to the constraints, babbles can detect insertion conflicts. On the contrary, we consider that two insertions never conflict. The main drawback of Babbles is the requirement of a primary site which have to resolve conflicts. This primary limits the scalability, and is a single point of failure.

The operational transformation (OT) approach [4] is composed by operations which express modifications, and the transformation functions used to modify concurrent operations toward local operations. A state vector is associated to each operations. The state vector size grows linearly with the number of sites and consequently, limits the scalability of this approach. The wooto approach depends only of the number of operations and not of the number of sites.

Distributed version control systems (DVCS) allow many users to edit the same documents concurrently. They provide the same features as CVS [18] or Subversion [19] without requiring a central site. DVCS do not express the notion of consistency. In addition, a well-known scenario from the OT approach [20] leads most of DVCS (Darcs, Mercurial, Git, Bazaar, ...) to the document inconsistency. Code Co-op [1] is a DVCS which allows to add a wiki to a replicated folder. Hence, we obtain a p2p wiki. As many DVCS, Code Co-op cannot ensure the wiki's pages consistency. Repliwiki is a P2P wiki based on the SHH-sync which is a DVCS. Unfortunately, as claimed before, DVCS algorithms failed to ensure consistency.

5 Conclusion

Wooki is a P2P wiki system. Compared to traditional centralized wikis, a P2P wiki system is fault-tolerant, improves global performances and allows user to work off-line. Wooki uses the wooto algorithm to manage consistency of copies. Wooto is an optimized version of the woot algorithm and ensures eventual consistency. Compared to other optimistic replication approaches, wooto is designed for P2P networks and is fully compatible with P2P network constraints.

In this paper, we described how to combine the wooto algorithm with an epidemic dissemination algorithm. This combination provides a reliable P2P wiki system that scales and tolerates the "churn" of P2P networks. It supports long-term disconnections without ever loosing off-line work. It also does not require any permanent sites, and the P2P network can be unstructured.

The wooki system has several open issues:

- We clearly make the choice to keep all tombstones in the wooto page model. It implies that wooki pages will grow infinitely. Nevertheless, keeping a tombstone has a very small space overhead. In the context of a wiki system, this choice is reasonable. In another context, it is interesting to design a distributed tombstone garbage collector that is compatible with P2P constraints.
- A P2P wiki system changes interactions between humans and the system. In a central wiki system, a user is aware about concurrent changes when he saves its page. If the wiki page has changed during the edition, the user have to resolve conflicts before committing his changes. In a P2P wiki system, a user can save its changes before concurrent ones arrive for integration. Wooto will solve the conflicts but the final page available on the site is a page computed by the system but not reviewed by a human. We need to add to Wooki an awareness engine that is responsible to warn about concurrent changes.
- Replicating a wiki site is not completely transparent. If a wiki page contains some macros that computes some statistics, the visible result will not be the same on all sites. Suppose a counter that counts the read number of a page. If the page is replicated, this counter can have different values on different sites. This does not violate eventual consistency. The source page is still the same on all sites, but the rendering of the page can be different.
- In collaborative editing, it is important to undo some operations. In traditional wikis, it is possible to revert to a previous version of the wiki. In the general case, it must be possible to undo any operations (not always the last one), anytime [21]. We have not yet provided such features in wooto. However, as we keep all informations within pages, it should be possible to undo an operation. We still need to design such an algorithm and prove eventual consistency in case of undo.
- In Wooki, we built a new P2P wiki system. We are working on integration of the wooki engine with existing wiki systems. The general approach is to manage optimistic replication by only using the web service interface of existing wiki systems.

References

1. Reliable Software: Code Co-op. (2006) http://www.relisoft.com/co_op.
2. Kang, B.B., Black, C.R., Aangi-Reddy, S., Masri, A.E.: Repliwiki: A next generation architecture for wikipedia. (Unpublished) http://isr.uncc.edu/repliwiki/repliwiki - conference.pdf
3. Saito, Y., Shapiro, M.: Optimistic replication. ACM Computing Surveys **37**(1) (2005) 42–81
4. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: SIGMOD Conference. Volume 18. (1989) 399–407
5. O'Brien, J., Shapiro, M.: An application framework for nomadic, collaborative applications. In: IFIP WG 6.1 International Conference, DAIS. Volume 4025 of LNCS., Bologna, Italy (June 2006) 48–63
6. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for P2P collaborative editing. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW, Banff, Alberta, Canada (November 2006)
7. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565
8. Spencer, H., Lawrence, D.: Managing Usenet. O'Reilly (January 1998)
9. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenkcr, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC. (1987) 1–12
10. Yu, Y., Manolios, P., Lamport, L.: Model checking TLA+ specifications. In: Proceedings of Correct Hardware Design and Verification Methods - CHARME'99. (1999) 54–66
11. Myers, E.W.: An o(nd) difference algorithm and its variations. Algorithmica **1**(2) (1986) 251–266
12. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Trans. Comput. Syst. **17**(2) (1999) 41–88
13. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM Trans. Comput. Syst. **21**(4) (2003) 341–374
14. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)
15. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
16. Birman, K.P., Joseph, T.A.: Exploiting virtual synchrony in distributed systems. In: Symposium on Operating Systems Principles (SOSP). (1987) 123–138
17. Kermarrec, A.M., Rowstron, A.I.T., Shapiro, M., Druschel, P.: The IceCube approach to the reconciliation of divergent replicas. In: Proceedings of the ACM symposium on Principles of distributed computing, PODC. (2001) 210–218
18. Berliner, B.: CVS II: Parallelizing software development. In: Proceedings of the USENIX Winter Technical Conference, Berkeley, California, USA (1990) 341–352
19. CollabNet, Inc.: Subversion. (2005) http://subversion.tigris.org/.
20. Oster, G., Urso, P., Molli, P., Imine, A.: Tombstone transformation functions for ensuring consistency in collaborative editing systems. In: The International Conference on Collaborative Computing, CollaborateCom, Atlanta, Georgia, USA, IEEE Press (November 2006)
21. Sun, C.: Undo as concurrent inverse in group editors. ACM Transactions on Computer-Human Interaction (TOCHI) **9**(4) (December 2002) 309–361