Performance Evaluation of a Privacy-Enhancing Framework for Personalized Websites[*]

Yang Wang and Alfred Kobsa

Donald Bren School of Information and Computer Sciences
University of California, Irvine, U.S.A.
{yangwang,kobsa}@ics.uci.edu

Abstract. Reconciling personalization with privacy has been a continuing interest in the user modeling community. In prior work, we proposed a dynamic privacy-enhancing user modeling framework based on a software product line architecture (PLA). Our system dynamically selects personalization methods during runtime that respect users' current privacy preferences as well as the prevailing privacy laws and regulations. One major concern about our approach is its performance since dynamic architectural reconfiguration during runtime is usually resource-intensive. In this paper, we describe four implementations of our system that vary two factors, and an in-depth performance evaluation thereof under realistic workload conditions. Our study shows that a customized version performs better than the original PLA implementation, that a multi-level caching mechanism improves both versions, and that the customized version with caching performs best. The average handling time per user session is less than 0.2 seconds for all versions except the original PLA implementation. Overall, our results demonstrate that with a reasonable number of networked hosts in a cloud computing environment, an internationally operating website can use our dynamic PLA-based user modeling approach to personalize their user services, and at the same time respect the individual privacy desires of their users as well as the privacy norms that may apply.

1 Introduction

Since personalized websites collect personal data, they are subject to prevailing privacy laws and regulations if the respective individuals are in principle identifiable (see [1] for a comprehensive review of privacy issues in personalization). Internationally operating websites are particularly affected since a large number of countries extend the applicability of their privacy laws to operators and personal data flows beyond their national boundaries. Moreover, in order to encourage users to interact with personalized sites and thus benefit from the full potential of personalization, personalized systems should also cater to each user's current privacy preferences. That is to say, a user can have varying privacy preferences on different sites, and at different times on the same site, and thus each site should be able to treat the same user differently depending on

[*] This research has been supported through NSF grant IIS 0308277 and a Google Research Award. We would like to thank Scott Hendrickson, Eric Dashofy, André van der Hoek and the UMAP09 reviewers for their helpful comments.

her current privacy preferences. In [2] we illustrated that these privacy constraints may affect not only the data that may be collected by the personalized website, but also the admissibility of personalization methods for processing personal data.

The resulting combinatorial complexity of these privacy constraints make them hard to cope with. We therefore proposed a novel approach based on software product line architecture that models the variability in both the privacy and personalization domains, and allows the configuration of the employed personalization methods to be dynamically tailored to each user at runtime, considering both the prevailing privacy norms and the user's current privacy preferences. This flexible approach not only helps address the complexity of building personalized systems, but also strongly supports their evolution: as new privacy and personalization concerns arise, they can be added to the product line architecture in a modular manner [3, 4].

One major concern about our approach is its performance since dynamic architectural reconfiguration during runtime is usually resource-intensive. Will it be practically possible to deploy such a dynamic system in a contemporary internationally operating website? In this paper, we describe four variant implementations of our system and an in-depth performance evaluation under realistic workload conditions. Our work stands in the tradition of similar attempts in the past to gauge the performance of user modeling tools through simulation experiments (e.g., [5–7]). It is however also substantially different from prior evaluations due to the fact that the workload is not induced by user requests (such as web page requests) or requests from software processes (such as user-adaptive applications or personalization methods), and that the aspired goal is not a user modeling tool that performs personalization tasks efficiently. Rather, the workload is induced by the initiation of new user sessions, and the goal is the efficient instantiation of user-modeling architectures that meet the privacy constraint of each individual user.

In the remainder of this paper, we will first briefly recap our privacy-enhancing user modeling framework in Section 2. We then describe the setup of our performance evaluation, such as the simulated parameters and workload, in Section 3. Thereafter, we present different implementations of our approach in Section 4, the performance evaluation of these implementations in Section 5, discussions of the results in Section 6, and conclusions in Section 7.

2 Our Privacy-Enhancing User Modeling Framework

In order to enable personalized web-based systems to respect users' individual privacy constraints, Kobsa [8] proposed a user modeling framework that encapsulates different personalization methods in individual components and, at any point during runtime, ensures that only those components that comply with current privacy constraints can be used. We adopted a Product Line Architecture (PLA) approach to implement this design. A PLA is an architectural representation for a set of related products. It includes core elements present in all product architectures, and variation points where variations exist among individual product architectures [9]. Each variation point is guarded by a Boolean expression that specifies the conditions under which an optional component should be included in a particular product architecture [10]. A particular product ar-

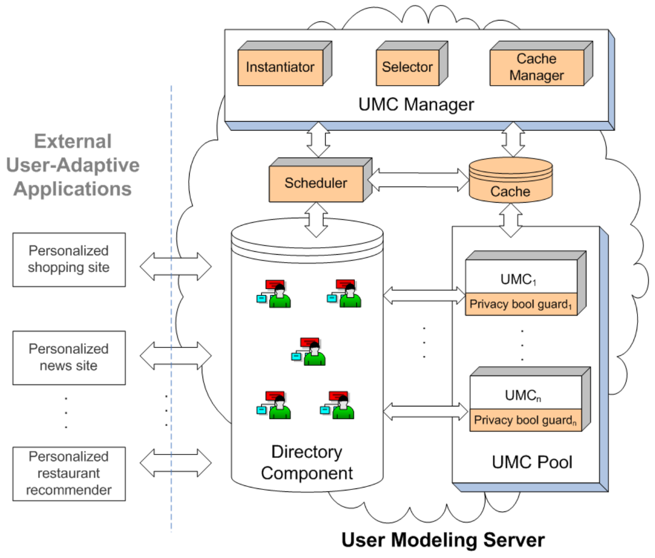

Fig. 1. Distributed dynamic privacy-enhancing user modeling framework

chitecture can be selected out of a product line architecture by resolving the Boolean guards of each variation point at design-time, invocation-time or run-time [11].

2.1 Framework Overview

Figure 1 shows an overview of our framework[1]. It consists of external user-adaptive applications, an LDAP-based user modeling server (UMS) [12], a user modeling component (UMC) manager, a Scheduler and a cache database. External user-adaptive applications can retrieve user information from the UMS so as to personalize services to their end users, and can submit additional user information to the UMS. The UMS includes a Directory Component and a pool of UMCs. The Directory Component hosts a repository of user models, storing users' characteristics and their individual privacy preferences. The UMC Pool contains a set of UMCs, each encapsulating one or more personalization methods (e.g., collaborative filtering). UMCs make inferences about users based on existing information in the user models and then add the derived user information to the user models [2].

To enable PLA operations (e.g., product architecture selection), the UMC Manager was added to the UMS. The enhanced UMS was then modeled as a PLA, in which the Directory Component and the UMC Manger were core components, and UMCs were optional components. Each UMC is guarded by a Boolean expression that represents privacy conditions under which the respective UMC may operate. Each privacy condition is expressed by a Boolean variable (e.g., Combining_Profile == true). As such,

[1] The shaded parts are our privacy-related additions to the user modeling server described in [12].

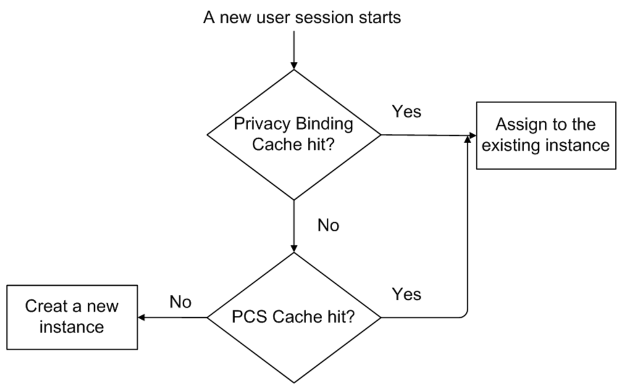
A new user session starts
Privacy Binding Cache hit?
Yes
Assign to the existing instance
No
PCS Cache hit?
Yes
No
Creat a new instance

Fig. 2. Multi-level caching mechanism

we use these Boolean variables bearing privacy semantics to represent users' privacy preferences as well as applicable privacy regulations. In practice, the values of these Boolean variables can come from the evaluation of privacy conditions expressed in a privacy policy language (see [13] for a discussion of these languages).

In the following, we will describe the UMC Manager in more detail and then discuss distribution issues.

2.2 UMC Manager

The UMC Manager was implemented to support PLA selection and instantiation as well as our caching mechanism. It consists of the following components:

Selector. When a new user session begins, the Selector takes the PLA and the privacy bindings relating to the new session as inputs. Privacy bindings are name-value pairs for the Boolean guards in the PLA, e.g., Combining_Profile = false which would represent that the user or some privacy norm relating to the user session disallow the merging of profiles relating to the same user. The Selector selects a particular product architecture out of the PLA by resolving the Boolean guards associated with each optional component in the PLA using the current privacy bindings. It expresses the chosen architecture through a binary Privacy Constraint Satisfaction (PCS) vector [3] whose nth element represents whether or not the nth UMC may be included in the selected product architecture.

Instantiator. The Instantiator takes a PCS as input and creates a runtime system instance for the product architecture. The total number of different PCS vectors ($2^{TotalUMCs}$) equals the theoretical maximum of instances that may be created.

Cache Manager. We designed a multi-level caching strategy that is shown in Fig. 2. The Cache Manager controls caches of both individual users' privacy bindings and their associate PCS vectors (i.e., the results of the PLA selection). More specifically, when a new user session starts, the Cache Manager checks the privacy binding cache whether the system has an existing user session with the same privacy bindings (i.e., a user with identical privacy norms and individual privacy preferences). If it finds one, the new session will be assigned to the same system instance as the existing session. If no such binding can be found, the Cache Manager will further check the PCS cache since a PCS may meet the constraints of more than one privacy binding. Only if no such PCS can be found either, the Instantiator will start a new instance for this user session. More details about our runtime dynamism mechanism can be found in [3].

2.3 Distributed Framework

In order to cope with potentially millions of concurrent users, the enhanced UMS needs to be distributed. In Fig. 1, the cloud denotes the distribution of processing over a network of machines. Distribution of the LDAP-based Directory Component and the UMC Pool have been addressed in [12]. We also distribute the UMC Manager over a network of hosts, each having a stand-alone copy of the UMC Manager. In addition, we add a Scheduler in the framework to assign incoming user sessions to various hosts, and a database to store the privacy binding cache and the PCS cache.

3 System Implementation

In this section, we describe the implementations of major components and operations in our framework (the first two were varied in the different conditions of our experiment).

3.1 PLA Representation, Selection and Instantiation

As explained above, our privacy-enhancing user modeling framework was designed as a PLA. Therefore, the core of the framework involves the following tasks: generation of a PLA for the system architecture, selection of UMCs based on the bindings of the privacy Boolean guards, and instantiation of the selected architecture for the user modeling system.

ArchStudio-based Implementation. In our preliminary implementation [3], we adapted functionalities from ArchStudio 3 [14] to perform the above tasks. ArchStudio 3 is an architecture-centric development environment, built on the C2 architectural style [15]. It provides excellent support for PLA modeling and development. This system has been meanwhile upgraded to ArchStudio 4 [16], built on the Myx architectural style [17]. The Myx style provides better system performance because it allows unmediated synchronous procedure calls between components in the architecture. In the C2 style, component interactions are always asynchronous and mediated by connectors. We therefore chose ArchStudio 4 for our final test system and implemented it in the Myx style (we call it the Myx version).

Our Customized Implementation. The standardization and extensibility of the XML-based PLA representation come at a price: XML processing can be expensive and thus affect the overall system performance. This is especially the case when the PLA has a large number of components. Therefore, we designed a light-weight alternative to the xADL 2.0 representation, called PLA Object Notation (PLAON). It contains an array of component objects. Each optional component object stores its privacy Boolean guard in an array, each element representing a privacy Boolean variable. Privacy bindings are in turn stored as a binary array, each element denoting the binding for a privacy Boolean variable. Our customized selector can then use the privacy binding array to resolve the Boolean guard array. Again the results of the selection will be a PCS vector, implemented as a binary array. Our customized instantiator reads from the PCS array to start components whose values in the PCS array are 1. Since our customized implementation represents the PLA semantics in a succinct object notation and omits any XML processing, we expect it to perform better than the original Myx-based implementation.

3.2 Multi-Level Caching

Caching is the other factor that we vary in our experiment. As described earlier, if two users have the same privacy bindings, or the same PCS vectors after selection, then they can share the same user modeling system instance. This reuse would save the system from performing unnecessary architectural selections and instantiations in such cases.

3.3 Resource-Aware Scheduling

Since hosts can have different hardware and networking characteristics in our distributed framework (e.g. different amounts of memory), the scheduler needs to take this heterogeneity into account, so as to optimize the overall system performance. When a host becomes available, it will connect and register itself with the Scheduler. The scheduler keeps track of all the registered hosts, their computing capabilities (right now we only consider the memory size), and the number of user sessions that each host is currently serving. When a new user session is initiated, the Scheduler first checks with the Cache Manager to see if any system instance can be reused for this session. If not, it would select the lightest-loaded host that can still handle this session with its resources. This resource-aware scheduling was used in all conditions of our experiment.

4 Experimental Design and Procedures

4.1 Controlled variables

Since we suspected that the XML-based Myx implementation described in Sect. 3.1 would perform poorly, we aimed at contrasting it with the two optimization methods described in Sect. 3.1 and 3.2 through the following 2-factorial design: (Myx vs. Customized) $\times$ (Non-caching vs. Caching).

4.2 Simulation parameters

Since we anticipated that a very large network of machines will be needed to handle real-world large-scale applications that was unavailable to us, we identified a reasonable number of 3000 maximum users per host in pre-trials and simulated such a single host

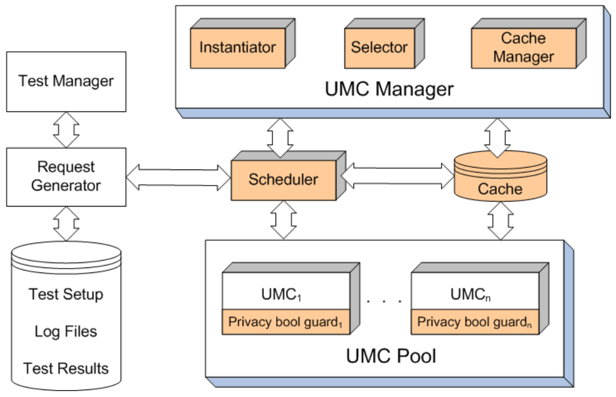

Fig. 3. Testbed architecture

on a PC. The other parameters of our experiment were chosen based on our analysis of international privacy laws and their impacts on personalized systems [18, 19, 2], as well as the user modeling literature:

- Total number of UMCs in the PLA: 10.
- Total number of different privacy constraints: 100.
- Simulated number of user sessions per host: 3000.
- Average arrival rate of unique visitors per host per second: 0.5.
- Number of variables in the privacy Boolean guards of each UMC: 5.

We randomly chose 5 out of the total 100 privacy constraints for each UMC and randomly generated the privacy bindings (true or false) for each user session.

Previous work such as [20, 21] has empirically shown that the arrival of new user sessions at a website largely follows a Poisson process[2]. To compare the four conditions of our experiment on a common basis, we pre-generated Poisson-distributed session arrival times with a mean rate of 0.5 users per second, and used them in all experiments.

4.3 Testbed

Figure 3 depicts the overall testbed architecture. The performance evaluation of the LDAP-based Directory Component and the UMC Pool in [12] had already demonstrated that they scale well and can be deployed to high-workload commercial applications. To be able to measure the performance of the PLA selection and instantiation in isolation, we omitted the Directory Component and created functionless dummy

[2] Chlebus and Brazier [21] found two separate regions of time in a day, each lasting several hours and having a different average arrival rate. They therefore suggests that the arrival rate rather follows a non-stationary Poisson process, i.e. consists of more than one Poisson process, each with its own rate. Those results are not likely to apply to internationally operating sites though on which we largely focus.

implementations for all UMCs, thereby realistically assuming that those components would run on different hosts anyways when deployed in practice. We added a Test Manager to control experiments, a Request Generator to generate user sessions, and a MySQL database to store the test setup, logs and results. The whole testbed except for the database was implemented in Java, complied in Java 1.6, and run in the HotSpot Java Virtual Machine on a PC platform with two 3.2 GHz processors, 3 GB of RAM, and a 150 GB hard disk.

4.4 Procedures

The Test Manager first reads the test setup from the database and informs the Request Generator to generate simulated user sessions and associated privacy bindings. The Request Generator reads the session arrival times from the database and starts sending user sessions to the Scheduler. The Scheduler chooses a host to handle the session. The host then performs the PLA selection and instantiation (in the Cache conditions, PLA selection and/or instantiation may be skipped, depending on the type of cache hit – see Sect. 2.2). Once the session has been assigned to a runtime system instance, the assignment is written into the cache if a cache is used. When all user sessions have been handled, log files and test results are written into the database.

For every user session, we measure three values:

Handling time, which is the period between the Request Generator sending the session to the Scheduler, and the session being assigned to a runtime instance.

Reuse rate of runtime instances, which considers the total number of user sessions and of instances currently in the system, has a range of [0, 1) and is calculated as $\frac{Total\ Sessions - Total\ Instances}{Total\ Sessions}$

Performance improvement (percentage), which compares the system performance of the original implementation (Myx implementation without caching) with that of an enhanced implementation. For a given number of users handled, this value has a range of [0, 1) and is calculated as $\frac{\sum TotalHandlingTimeOriginalVersion - \sum TotalHandlingTimeEnhancedVersion}{\sum TotalHandlingTimeOriginalVersion}$

5 Evaluation Results

5.1 Handling Time per User Session

Figure 4 plots the handling times for each user session in the four implementations, and indicates the means and standard deviations. We can see that the customized versions perform better than the Myx versions, that our multi-level caching mechanism improves both versions, and that the customized version with caching performs best. The average handling time per user session is less than 0.2 seconds for all versions except the Myx implementation without caching.

We also analyzed the spikes of the handling time in Fig. 4 and disconfirmed that they were correlated with bursts in the arrival rate. Based on an analysis of the logs created by our experimental testbed we found that the main reason for the delay lies

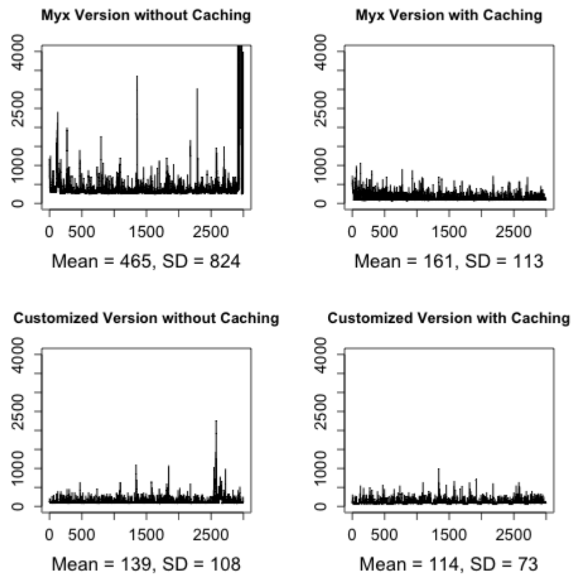

Fig. 4. Handling time for each user session (milliseconds)

in Java's indeterministic thread scheduling. Requests to handle a new session, select an architecture, and instantiate an architecture each creates a new thread, and occasionally one of the threads gets switched out of processing and later switched back in. One can notice that in the Myx version without caching, high handling times increase towards the end of the experiment. This is because the machine almost ran out of heap space, and the Java Virtual Machine kept switching threads. A good remedy for these effects of indeterministic thread switching is to shorten the processing time, which is confirmed by the substantial decrease of such delays in the conditions in which the customized version and/or caching have been used.

5.2 Runtime Instance Reuse Rate

Figure 5(a) plots the runtime instance reuse rates for the two caching versions (in the non-caching versions, no instances are being reused). The reuse rates for the caching versions increase degressively as the cumulative number of user sessions increases. The two curves are very similar because both versions use the same caching scheme; the small variations are due to the true randomness of privacy Boolean guard and privacy binding generation.

5.3 Performance Improvement

Figure 5(b) plots the performance gain of our three improved versions in comparison to the baseline Myx version without caching. The curve at the bottom (gain from Myx

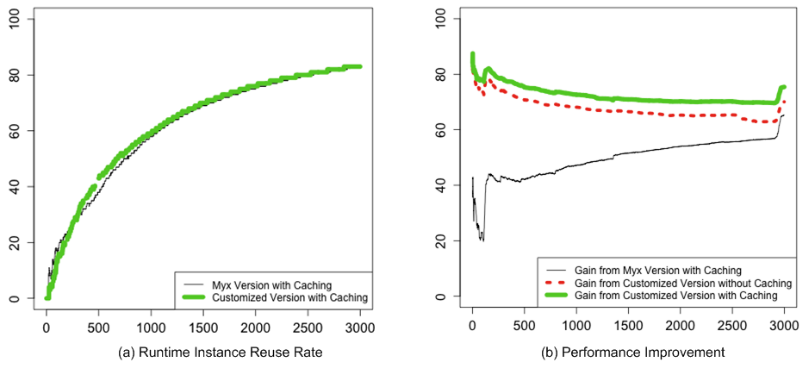

Fig. 5. Instance reuse and performance improvement (both in %), by cumulative number of users

version with caching) goes up as expected: the cache size increases with an increasing number of users, and hence the hit rate and thus the performance gain increase. The curve in the middle (gain from customized version without caching) is always above the first curve, meaning that the gains through customization are larger than through caching. As expected, this difference becomes smaller with increasing number of users and thus cache hits. The topmost curve shows the gains from both caching and customization. While the combined effect is always higher than each single effect, it is unfortunately not additive. While with increased number of users the gains through caching increase, each hit ”cancels out” the gains through customization which will not be invoked in such a case. Larger cache sizes still cause performance gains as is demonstrated by the slightly increasing distance between the middle and upper curve. This differential however grows far less than the slope of the lowermost curve which represents the gains through caching for the non-customized Myx version.

6 Discussion

Performance Improvement. The evaluation results show that both our customization and caching improve the performance. The customized versions use a light-weight PLA representation, which consumes less memory and enables faster PLA selection and instantiation than the XML-based Myx versions. The multi-level caching mechanism saves time and resources that would otherwise be spent on creating new runtime instances. Under the current completely random assignment of privacy guards and bindings, the probability of a privacy binding cache hit is $1/\ 2^{TotalConstraints}$(about 7.9e-31), while the probability of a PCS cache hit is $1/\ 2^{TotalUMCs}$ (about 9.8e-4). Therefore, the vast majority of instance reuses came from the PCS cache hits.

Practical Implications. The average arrival rate of new visitors in the current experiment setup is 0.5. In contrast, Yahoo.com which Alexa currently ranks No. 1 worldwide in terms of traffic seems to have a daily reach of close to 30 million unique visitors [22].

This roughly translates into an average arrival rate of 350 users per second. Because of its modular approach, our framework would be able to handle this workload in a cloud-computing paradigm [23]. If we continue using our average arrival rate of 0.5 visitors for each node, then we can handle Yahoo-sized traffic with a cloud that consists of 700 nodes on average. Therefore we believe that with sufficient support from a cloud computing environment, our approach can scale well to serve internationally operating websites, which would profit most from our privacy-enhancing framework. As a reminder though, this number does not include the nodes that would be required to run the Directory Component, the User Modeling Component, and the Web server.

Limitations of the Evaluation. Privacy bindings are randomly assigned to sessions in our simulation, and hence their variations are evenly distributed across users. In reality though, users' individual privacy preferences are likely to gravitate towards typical preferences, countries may have typical combinations of privacy bindings, and visitors from certain countries may be more frequent than from others. The hit rate in the privacy binding cache is likely to be higher in this more realistic scenario with uneven distribution, and the number of generated different instances lower than in our simulation, both of which reduces the memory load. Another limitation is that the experiments were conducted on a single PC platform. When the user modeling server is distributed in a cloud computing environment, the Scheduler and the cache database are likely to be overloaded, and therefore will need to be distributed as well.

7 Conclusions

Reconciling privacy and personalization in internationally operating websites is a challenging problem that no other existing work seems to address. Our PLA-based approach is aimed at filling this gap, but its resource-intensive PLA selection and instantiation process put the overall system performance into question. In this paper we discussed four implementations of our approach and evaluated their performance in a simulation experiment. Our study shows that our light-weight customized implementation performs better than the original PLA implementation (the Myx version), that our multi-level caching mechanism improves both versions, and that the customized version with caching performs best. The average handling time per user session is less than 0.2 seconds for all versions except the Myx version. Overall, our results demonstrate that with a reasonable number of networked hosts in a cloud computing environment, an internationally operating website can use our dynamic PLA-based user modeling approach to personalize their user services and at the same time respect the individual privacy desires of their users as well as the applicable privacy norms.

References

1. Kobsa, A.: Privacy-enhanced web personalization. In Brusilovsky, P., Kobsa, A., Nejdl, W., eds.: The Adaptive Web: Methods and Strategies of Web Personalization. Springer-Verlag (2007) 628–670

2. Wang, Y., Kobsa, A.: Respecting users' individual privacy constraints in web personalization. In Conati, C., McCoy, K.F., Paliouras, G., eds.: User Modeling 2007: 11th Intl. Conf., Springer (2007) 157–166
3. Wang, Y., Kobsa, A., van der Hoek, A., White, J.: PLA-based runtime dynamism in support of privacy-enhanced web personalization. In: SPLC 2006, IEEE Press (2006) 151–162
4. Wang, Y., Hendrickson, S.A., van der Hoek, A., Taylor, R.N., Kobsa, A.: Modeling PLA variation of privacy-enhancing personalized systems. Submitted for publication (2009)
5. Kobsa, A., Fink, J.: Performance evaluation of user modeling servers under real-world workload conditions. In Brusilovsky, P., Corbett, A.T., Rosis, F.d., eds.: User Modeling 2003: 9th Intl. Conf., Springer Verlag (2003) 143–153
6. Carmichael, D.J., Kay, J., Kummerfeld, B.: Consistent modeling of users, devices and sensors in a ubiquitous computing environment. User Modeling and User-Adapted Interaction **15**(3-4) (2005) 197–234
7. Zadorozhny, V., Yudelson, M., Brusilovsky, P.: A framework for performance evaluation of user modeling servers for web applications. Web Intelli. and Agent Sys. **6**(2) (2008) 175–191
8. Kobsa, A.: A component architecture for dynamically managing privacy constraints in personalized web-based systems. In Dingledine, R., ed.: Privacy Enhancing Technologies: 3rd Intl. Workshop, PET 2003. Springer Verlag (2003) 177–188
9. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, New York, New York (2002)
10. van der Hoek, A., Rakic, M., Roshandel, R., Medvidovic, N.: Taming architectural evolution. In: 9th ACM Symp. on the Foundations of Softw. Eng. (2001) 1–10
11. van der Hoek, A.: Design-time product line architectures for any-time variability. Sci. Comp. Prog., special issue on Softw. Variability Mgmt. **53**(30) (2004) 285–304
12. Kobsa, A., Fink, J.: An LDAP-based user modeling server and its evaluation. User Modeling and User-Adapted Interaction **16**(2) (2006) 129–169
13. Wang, Y., Kobsa, A.: Privacy-enhancing technologies. In Gupta, M., Sharman, R., eds.: Social and Organizational Liabilities in Information Security. IGI Global (2009) 203–227
14. ArchStudio: Archstudio 3. `www.isr.uci.edu/projects/archstudio/` (2005)
15. Taylor, R.N., e.a.: A component- and message-based architectural style for GUI software. IEEE Trans. Softw. Eng. **22**(6) (1996) 390–406
16. Dashofy, E., Asuncion, H., Hendrickson, S., Suryanarayana, G., Georgas, J., Taylor, R.: Archstudio 4: An architecture-based meta-modeling environment. In: ICSE 2007: Intl. Conf. on Softw. Eng., IEEE Computer Society (2007) 67–68
17. ArchStudio: Myx. `www.isr.uci.edu/projects/archstudio/myx.html` (2008)
18. Wang, Y., Chen, Z., Kobsa, A.: A collection and systematization of international privacy laws, with special consideration of internationally operating personalized websites. `http://www.ics.uci.edu/~kobsa/privacy` (2006)
19. Wang, Y., Kobsa, A.: Impacts of privacy laws and regulations on personalized systems. In Kobsa, A., Chellappa, R., Spiekermann, S., eds.: Proceedings of PEP06, CHI 2006 Workshop on Privacy-Enhanced Personalization. ACM (2006) 44–46
20. Bhole, Y., Popescu, A.: Measurement and analysis of HTTP traffic. Journal of Network and Systems Management (2005)
21. Chlebus, E., Brazier, J.: Nonstationary poisson modeling of web browsing session arrivals. Information Processing Letters **102**(5) (2007) 187–190
22. Alexa: Yahoo traffic details. `http://www.alexa.com/data/details/traffic_details/yahoo.com` (2009)
23. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: 10th IEEE Intl. Conf. on High Perf. Comp. and Comms., IEEE Computer Society (2008) 5–13