Transactional Consistency in User Modeling Systems

Josef Fink

GMD FIT, German National Research Center for Information Technology,
Sankt Augustin, Germany

Abstract. It is widely accepted that the consistency of adaptive interfaces is crucial for their usability. Many threats for consistency in adaptive applications have been reported in the literature so far (e.g., consistency of adaptation methods and techniques, consistency of the user model). In this paper we argue that many, if not all, user modeling systems that have been developed so far are substantially threatening consistency by offering no adequate means for communicating consistency contexts. This is especially the case for user modeling servers, which are supposed to serve several applications in parallel. In order to prevent consistency problems in user modeling systems, we introduce basic concepts and techniques from transaction management. User modeling systems that adhere to the principles of transaction management can be expected to provide a reliable source of user information for adaptive applications, especially in real world settings.

Introduction

One of the central tasks of a user modeling system is the provision of user-related information to one or more applications for adaptation purposes. An interface specifies the externally available functionality of the user modeling system and an associated protocol which controls the communication between a user modeling system and one or more adaptive applications. Several interfaces and communication protocols have been proposed in the user modeling literature so far (e.g., Finin, 1989; Brajnik and Tasso, 1994; Paiva and Self, 1995; Kay, 1995; Kobsa and Pohl, 1995; Orwant, 1995; Kobsa et al., 1996; Brusilovsky et al., 1997; Kummerfeld and Kay, 1997). Despite of their differences, most, if not all, have one feature in common: Each access to a user model is treated as a request that is discrete, i.e. independent from former and subsequent accesses. The sophisticated internal functionality offered by many user modeling systems (e.g., activation and deactivation of stereotypes, drawing of inferences, preservation of model consistency and integrity) is executed autonomously and is isolated from external accesses.

This isolation practice is in sharp contrast to the cohesion between subsequent user model accesses, which can be found in many applications. Some examples from the area of adaptive hypermedia which motivate the cohesion between subsequent queries to a user model on the level of adaptations are:

– a single adaptation is triggered by several queries about a user's presumed knowledge (e.g., if one of two concepts are presumed to be known by the user then automatically provide a comparison of the two concepts (adapted from Brusilovsky, 1996));

– an adaptation is triggered by several queries about a user’s presumed interests several times on a hypermedia page (e.g., if the user is presumed to be interested in historical details then automatically provide links to historical information for all points of interest (Fink et al., 1997));
– several adaptations on related hypermedia pages are triggered by several queries about a user’s presumed knowledge about concepts (e.g., according to the user’s presumed knowledge about the underlying concepts of a curriculum, use a traffic light metaphor on one page and, connected via a NEXT button, suggest an individual curriculum sequencing on a second page (Weber and Specht, 1997)).

The cohesion between subsequent user model accesses stems from the associated consistency context of an adaptation. Following the examples mentioned above, a consistency context can comprise of a single adaptation, several adaptations on a single hypermedia page, or several adaptations on a set of related hypermedia pages. An adaptive system needs to communicate this context to a user modeling system in order to be able to adapt in a consistent manner. The user modeling system in turn, is expected to preserve the cohesion between user model accesses by providing a unique ”sphere of control” (Gray and Reuter, 1993). This means basically that internal functionality of the user modeling system and accesses from other applications that read and update the same parts of the user model are not concurrently executed.

If functionality for communicating and preserving consistency contexts is lacking, as it is the case in many user modeling systems today, then the consistency of adaptive applications is endangered. Let’s exemplify this for the second example presented above. Assume that the user modeling system is queried several times for the user’s presumed interest in historical information for points of interest (e.g., churches, museums, public places). The user modeling system, unaware of the coherence between subsequent calls for the user’s interest, activates internal functionality (e.g., an inference) that alters this presumed interest after the first query. Subsequent queries now return a different result, leading to inconsistent adaptations on a single hypermedia page (e.g., a link to historical information is only provided for the first point of interest and not for the other points of interest or vice versa).

So far, we argued that the loss of cohesion between user model accesses can lead to consistency problems in adaptive applications. In the following chapter, we will further elaborate this with a special emphasis on related problems in user modeling systems. All user modeling systems, regardless of their architecture (e.g., embedded within the application, client/server), that offer internal user modeling functionality and at the same time no means for communicating and preserving consistency contexts cannot guarantee consistency. This is especially the case for user modeling servers which are intended to serve several applications in parallel.

In order to avoid these types of consistency problems, we introduce in Chapter 3 concepts and basic techniques from transaction management. Transactions are introduced as indivisible units of work that are used for communicating consistency contexts to a user modeling system. In Chapter 4 we summarize our findings, discuss related work, and propose directions for future developments.

Consistency Problems

This chapter discusses consistency problems that can occur in user modeling systems and related applications when at least two unsynchronized sequences of logically related operations (threads) that belong to different consistency contexts operate simultaneously on the same parts of a user model. Threads can be incorporated in adaptive applications and user modeling systems (e.g., graphical interfaces of the user modeling system, activation and deactivation of stereotypes, drawing of inferences, acquisition of assumptions based on user's behavior).

The following examples describe two threads that simultaneously access the same user model by calling the functions ”ReadUM” and ”WriteUM”:

– ReadUM takes as an argument an assumption and returns yes or no, thereby indicating whether the assumption or its negation is supported by the user model. ”nil” is returned by ReadUM in case of the assumption nor its negation being supported by the user model. ReadUM(KnowsMonaLisa), for example, queries the user model for the user's presumed knowledge of the concept ”MonaLisa”.
– WriteUM takes two arguments: The first one is the name of the assumption that is to be inserted into the user model and the second one is the associated boolean value (i.e., yes or no). WriteUM(KnowsMonaLisa, yes), for example, updates the user model with the user's presumed knowledge of the concept ”MonaLisa”.

Application functionality and adaptations have been widely omitted for brevity in the following examples. Local thread variables are labeled ”X” and ”Z”. The entries in the user model reflect the state after the execution of corresponding thread functionality.

We will discuss the consistency problems ”inconsistent analysis”, ”dirty read”, and ”lost update” (adapted from Saake et al., 1997). These are so-called multi-transaction anomalies (or multi-user anomalies) in the area of database and transaction management (Bernstein et al., 1987).

1.1 Inconsistent Analysis

Each of the following threads strictly preserves consistency when exclusively accessing a user model. But inconsistencies can arise if these threads run interlocked. An example of such a situation is depicted in Table 1. Thread 1, which is incorporated in an adaptive hypermedia system, queries the user modeling system for the user's presumed knowledge of the two concepts ”MonaLisa” and ”CharlesI”. If the user model supports ”KnowsMonaLisa” or ”KnowsCharlesI” then a comparison of the two concepts is inserted on the current hypermedia page (this is done by calling ”ProvComp(MonaLisa, CharlesI)”). Thread 2 is issued by an asynchronously running acquisition component that deduces from user's answers to a quiz that he can be presumed to know ”MonaLisa”, but not ”CharlesI”.

Table 1. Inconsistent analysis.

Threads		User Model	
Thread 1	Thread 2	Knows-MonaLisa	Knows-CharlesI
X = ReadUM(KnowsMonaLisa)		no	yes
	WriteUM(KnowsMonaLisa, yes)	yes	yes
	WriteUM(KnowsCharlesI, no)	yes	no
Z = ReadUM(KnowsCharlesI)		yes	no
if ((X = yes) or (Z = yes)) 💣 ProvComp(MonaLisa, CharlesI)		yes	no

Because both variables X and Z contain no, the *comparison* of ”MonaLisa” and ”CharlesI” *is not provided* by the adaptive hypermedia system. Despite the fact that each thread successfully completed its work and the user model would have supported the provision of the adaptation in its initial and its final state. Thread 2 updates an assumption in the user model that Thread 1, which is still active, has previously read. After Thread 1 resumes its work, the content of X doesn’t reflect the current state of the user model anymore. Therefore, all further work that is controlled by X is probably inconsistent as well.

However, the comparison is correctly presented if these threads are sequenced by the transaction manager of the user modeling system (i.e., Thread 2 is run after Thread 1 has concluded its work). Transactions and associated components for user modeling systems will be discussed in Chapter 3.

1.2 Dirty Read

A ”dirty read” is issued if the information to be read has been previously updated by another, not yet successfully completed thread. If this thread has to be rolled back for whatever reason, for example, due to a user cancellation or a system crash, then the consistency can be endangered.

An example of such an interlocked situation at running time is depicted in Table 2. Thread 1, which belongs to an adaptive hypermedia system, observed that the user requested for historical information about the ”Louvre” (via a call to ”ReqForHistInf(Louvre)”) and updates the user model with the user’s presumed interest in historical information accordingly (abbr. ”IntHist”). Afterwards, the user cancels this thread, thereby initiating a rollback of all updates made to the user model so far. Thread 2 implements a simple inference rule within the user modeling system: If the user is presumed to be interested in historical information then he can also be presumed to be interested in art history (abbr. ”IntArtHist”).

Table 2. Dirty read.

Threads		User Model	
Thread 1	Thread 2	IntHist	IntArtHist
		no	no
if (ReqForHistInf(Louvre)) WriteUM(IntHist, yes)		yes	no
	Z = ReadUM(IntHist)	yes	no
	if (Z = yes) WriteUM(IntArtHist, yes)	yes	yes
Rollback	💣	no	yes

The *update issued by Thread 2 is based on invalid information* previously read from the user model. In addition, since Thread 2 already concluded its work, this update can’t be rolled back by the user modeling system anymore. Therefore, it remains persistent in the user model and is a potential source for arbitrary inconsistencies in the future.

However, consistency is preserved if these transactions are sequenced by the transaction manager of the user modeling system (i.e., Thread 2 is run after Thread 1 has been rolled back by the user modeling system).

1.3 Lost Update

Updates can get lost if at least two threads are concurrently updating the same information in a user model. In our example, Thread 1, which belongs to an asynchronously running news filtering system, queries the user modeling system for the user’s presumed interest in historical information. After some work, it updates the user model with the same information previously found in the user model and concludes. Likewise, Thread 2, which belongs to a user model inspection interface, queries the user modeling system for the user’s presumed interest in historical information and displays the result. The user changes this presumed interest and finally updates the user model.

Consistency is violated from a user’s point of view, if we consider for Thread 1 and 2 an interlocked situation at running time like the one depicted in Table 3:

Table 3. Lost update.

Threads		User Model
Thread 1	Thread 2	IntHist
X = ReadUM(IntHist)		yes
	Z = ReadUM(IntHist)	yes

	/* Display Z, user sets Z to no. */
	if (Z <> nil)
	  WriteUM(IntHist, Z)
		no

if (X <> nil)
  WriteUM(IntHist, X) 💣
		yes

The *update* the user initiated *gets lost* without any notification from the user modeling system. Despite the fact that both threads committed their work successfully, the update issued by Thread 2 is overwritten by Thread 1.

Like in the previous examples, consistency is preserved if these threads are sequenced by the transaction manager of the user modeling system (i.e., Thread 2 is run after Thread 1 ended).

Transactions

In Chapter 3.1, we introduce transactions and their properties against the background of the consistency problems we pointed out in the previous chapter. Based on that, we briefly discuss how transactions can be implemented in user modeling systems. In Chapter 3.2, we present the model of a ”flat transaction” and motivate the need for more sophisticated transaction models in user modeling systems.

1.4 Transaction Properties

Transactions have been introduced in the areas of database and transaction management in order to relieve an application programmer from all aspects of concurrency and failure. A transaction is a set of logically related operations that adheres to the classical *”ACID” principle* (Gray and Reuter, 1993; Orfali et al., 1994). Against the background of user modeling, the different properties of a transaction can be described as follows:

– ***A***tomicity means that a transaction is an indivisible unit of work: Either all or no accesses are processed by the user modeling system.
– ***C***onsistency means that a transaction transforms a user model from one consistent state into another consistent state. If such a state can’t be achieved (e.g., because of integrity constraints being violated), the user model has to be reset to the state before the transaction started.

– ***I***solation means that a transaction (e.g., initiated by an adaptive application) is not affected by other, concurrently executed transactions (e.g., drawing of inferences within the user modeling system, simultaneous access of several adaptive applications) with respect to shared parts of the user model. Changes initiated by a transaction are not visible to other transactions until this transaction has successfully ended.
– ***D***urability means that once a transaction has been successfully completed all changes made to the user model are persistent, i.e. these changes survive system failures.

In database and transaction management systems, a *transaction manager* supervises the progress and state of a transaction (Gray and Reuter, 1993; Saake et al., 1997). In order to achieve this, it interacts closely with a *scheduler* that controls the relative order in which concurrent operations are executed. The overall aim of the scheduler is to maximize potential concurrency, yet preventing consistency problems that may arise from several transactions running in parallel (see Chapter 2). The synchronization of transactions preserves the properties isolation and consistency. Synchronization can be implemented by various locking strategies on data elements and associated protocols. The preservation of the transaction properties atomicity and durability is the main aim of the *recovery manager*. Its tasks include all aspects of transaction commitment and abortion including the rollback to a consistent database state after a system crash.

In order to offer transactions that adhere to the ACID principle, user modeling systems have to incorporate a transaction manager, a scheduler, and a recovery manager. The sophistication of their implementation however, depends heavily on the intended deployment scenario and the services offered. If the user modeling system is embedded in a single-user application then various internal functionality (e.g., activation of stereotypes) has to be mainly synchronized with transactions issued by the application. The scheduler and the recovery manager can be implemented in this scenario in a very rudimentary form. Synchronization can be enforced for example via one or more locks (e.g., separate locks for read and write operations) on the whole user model and recovery can be implemented by managing all user model updates in a persistent buffer until a transaction successfully commits. User modeling server however, need to implement much more sophisticated techniques in order to synchronize various internal functionality with accesses from several applications. The aforementioned lock granularity may serve as an example. It is hardly desirable that a single transaction locks the whole user model, thereby preventing all other transactions using the same model from being executed, when operating itself on a relatively small part of the user model. More fine-grained locks (e.g., on an interest partition of the user model, on an assumption about a user's interest) increase potential concurrency but raise at the same time the need for a dedicated locking manager. Its main tasks are to synchronize locking by enforcing a specific protocol (e.g., the two phase locking protocol) and handle problems that may arise from transactions incompatible locking behavior (e.g. deadlocks). For further information on this subject we refer to Gray and Reuter (1993).

From an implementation point of view, transaction management components will not have to be developed from scratch. If a database management system that offers transaction support is used as a basis for example, then the user modeling system can probably take advantage of the already available functionality.

1.5 Transaction Models

The examples presented in Chapter 2 can be modeled as ”flat transactions” in the sense that all operations within a thread are on the same level and belong to the same consistency context. Flat transactions either succeed, or don’t happen at all. Typically, they run only a few seconds in order to occupy shared resources (e.g., user model contents) as less as possible. The major advantages of flat transactions are simplicity and ease of use. Against the background of user modeling and user-adaptive applications, flat transactions seem to cover many basic consistency requirements, for example on the level of adaptations. However, there are scenarios that point out the limitations of flat transactions in general (Orfali et al., 1994; Saake et al., 1997), which are also relevant for user modeling systems:

– Transactions with humans in between:
  As a simple example, we can take a component of a user modeling system that allows a user to inspect and edit his user model (see Chapter 2.3). If such a tool is designed as a single flat transaction, then the whole user model is locked until the user exits. No adaptive application and no internal functionality of the user modeling system that accesses the same parts of the user model can run in the meantime. In order to avoid this, the flat transaction can be split into a query transaction and one or more update transactions. Before updating the user model, each update transaction would have to revalidate the user model contents first, before initiating the update.
– Transactions that need to be partially rolled back:
  An example can be a user-adaptive shop based on the World-Wide Web, where everything between two page requests is being modeled as a single flat transaction. If the user cancels for example the shopping transaction, then all updates to the database, the user model, and a component for learning from navigation behavior will have to be rolled back for the page of concern. However, only the rollback of the database updates and user model updates is mandatory, the rollback of the learning component updates is optional.
– Transactions that span over a long time:
  We can take a user modeling server as an example that temporarily replicates the user model or parts of it to a mobile user agent. Eventually, updated versions of the agent’s user model have to be resynchronized with the server’s version at a later point in time. Again, designing the whole process as a single flat transaction makes little sense, if any. The replicated parts of the central user model would be locked for the whole duration of the transaction and if any problems would occur during resynchronization, the whole transaction would have to be rolled back.

Concluding, the rigidity of ACID-based flat transactions is quite inappropriate in scenarios where flexibility is needed. Up to now, various alternatives to flat transactions have been proposed. Most of them extend the linear flow of control in flat transactions by either linearly chaining units of work (e.g., ”chained transactions”) or by creating nested hierarchies of units of work (e.g., ”nested transactions”). More recently, sophisticated transactional models have also been proposed for long-living transactions (e.g., ”ConTracts”, ”migrating transactions”, ”shopping cart transactions”). The discussion of these transaction models and their reflection against the background of user modeling systems however, goes beyond the scope of this paper.

Discussion

The aim of this paper was to point out potential consistency problems in adaptive applications and user modeling systems that stem from unsynchronized accesses to shared parts of a user model. In order to avoid this, consistency contexts have to be communicated by adaptive applications and internal functionality of a user modeling system. The application interface of user modeling systems should provide commands like BeginOfTransaction, EndOfTransaction, and RollbackTransaction. A consistency context is communicated by BeginOfTransaction and EndOfTransaction, whereas the rollback of a user model to the state before the transaction started is communicated by RollbackTransaction.

To the best of our knowledge, none of the interfaces and communication protocols that have been proposed in the user modeling literature so far (e.g., Finin, 1989; Brajnik and Tasso, 1994; Paiva and Self, 1995; Kay, 1995; Kobsa and Pohl, 1995; Orwant, 1995; Kobsa et al., 1996; Brusilovsky et al., 1997; Kummerfeld and Kay, 1997) puts transactional facilities and associated ACID properties at the disposal of the application developer. This is quite amazing, given the potential consistency problems pointed out in this paper and the paramount importance of consistency for the overall usability of applications (Shneiderman, 1987; Nielsen, 1993; ISO, 1995; ISO, 1996).

A static and restricted form of transaction support that focuses on the ACID properties consistency and isolation can be found in a number of systems including the PAT-InterBook system (Brusilovsky et al., 1997) and the LDAP (Lightweight Directory Access Protocol) protocol (Wahl et al., 1997), which has been recently proposed for user modeling purposes by Kummerfeld and Kay (1997). These systems offer applications the opportunity to communicate user model accesses that belong to a consistency context in a package, which, in turn, is handled by these systems in a consistent and isolated manner. Although useful in many adaptation settings, this approach is rather restricted because of the ACID properties atomicity and durability being not provided and static because of the scope of a consistency context being limited to a single package. More complex consistency contexts (e.g., adaptations on a set of related hypermedia pages) can hardly be covered by such an approach without placing considerable burden on programmers of adaptive applications.

In order to preserve consistency, internal functionality of a user modeling system that is operating on shared parts of a user model has to communicate consistency contexts as well. Examples of such internal functionality can be found in user modeling servers (e.g., activation and deactivation of stereotypes in BGP-MS (Kobsa and Pohl, 1995), integration of results from (machine) learning techniques in Doppelgänger (Orwant, 1995)). To the best of our knowledge, these user modeling servers do not communicate consistency contexts in their internal functionality.

The research area of transaction management provides a rich arsenal of concepts, techniques, and implementations that can help developers of user modeling systems in providing transactional properties and developers of adaptive systems in designing consistent applications. User modeling systems that adhere to the principles of transaction management can be expected to provide a reliable source of user information, especially in real world settings.

References

Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Reading , MA: Addison-Wesley.

Brajnik, G., and Tasso, C. (1994). A shell for developing non-monotonic user modeling systems. *International Journal of Human-Computer Studies* 40:31-62.

Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction* 4(2):59-106.

Brusilovsky, P., Ritter, S., and Schwarz, E. (1997). Distributed intelligent tutoring on the Web. In du Boulay, B., and Mizoguchi, R., eds., *Proceedings of AI-ED'97*. Amsterdam: IOS. 482-489.

Finin, T. W. (1989). GUMS: A general user modeling shell. In Kobsa, A., and Wahlster, W., eds., *User Models in Dialog Systems*. Berlin, Heidelberg: Springer. 411-430.

Fink, J., Kobsa, A., and Nill, A. (1997). Adaptable and adaptive information access for all users, including the disabled and the elderly. In Jameson, A., Paris, C., and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference*. Wien, New York: Springer. 171-173.

Gray, J., and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann.

ISO. (1995). *Ergonomic Requirements for Office Work with Visual Display Terminals, Part13, User guidance*. International Standard.

ISO. (1996). *Ergonomic Requirements for Office Work with Visual Display Terminals, Part12, Ergonomic requirements for presentation of information*. Draft International Standard.

Kay, J. (1995). The um toolkit for reusable, long term user models. *User Modeling and User-Adapted Interaction* 4(3):149-196.

Kobsa, A., and Pohl, W. (1995). The user modeling shell system BGP-MS. *User Modeling and User-Adapted Interaction* 4(2):59-106.

Kobsa, A., Fink, J., and Pohl, W. (1996). *A Standard for the Performatives in the Communication between Applications and User Modeling Systems (draft)*. Available at http://zeus.gmd.de/~kobsa/rfc.ps

Kummerfeld, R., and Kay, J. (1997). Remote access protocols for user modelling. In *Proceedings and Resource kit for Workshop User Models in the Real World*. Chia Laguna, Sardinia. 12-15.

Nielsen, J. (1993). *Usability Engineering*. San Diego, CA: Academic Press.

Orfali, R., Harkey, D., and Edwards, J. (1994). *Essential Client/Server Survival Guide*. New York, Singapore: Wiley & Sons.

Orwant, J. (1995). Heterogeneous learning in the Doppelgänger user modeling system. *User Modeling and User-Adapted Interaction* 4(2):107-130.

Paiva, A., and Self, J. (1995). TAGUS-A user and learner modeling workbench. *User Modeling and User-Adapted Interaction* 4(3):197-226.

Saake, G., Schmitt, I., and Türker, C. (1997). *Objektdatenbanken – Konzepte, Sprachen, Architekturen*. Bonn, London: Thomson.

Shneiderman, B. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. New York, Tokyo: Addison-Wesley.

Wahl, M., Howes, T., and Kille, S. (1997). *Lightweight Directory Access Protocol (v3)*. Available at ftp://ftp.ietf.org/internet-drafts/draft-ietf-asid-ldapv3-protocol-09.tx

Weber, G., and Specht, M. (1997). User modeling and adaptive navigation support in WWW-based tutoring systems. In Jameson, A., Paris, C., and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference*. Wien, New York: Springer. 289-300.