Personal Lifelong User Model Clouds

Peter Dolog[1], Judy Kay[2], and Bob Kummerfeld[2]

[1] Aalborg University, Department of Computer Science,
Selma Lagerlöfs Vej 300,DK-9220 Aalborg East, Denmark
dolog@cs.aau.dk

[2] University of Sydney, School of Information Technologies,
Level 3 West, Building J12, NSW, 2006, Australia
{judy,bob}@it.usyd.edu.au

Abstract. This paper explores an architecture for very long term user modelling, based upon *personal user model clouds*. These ensure that the individual's applications can access their model whenever it is needed. At the same time, the user can control the use of their user model. So, they can ensure it is accessed only when and where they wish, by applications that they wish. We consider the challenges of representing user models so that they can be reused by multiple applications. We indicate potential synergies between distributed and centralised user modelling architectures, proposing an architecture which combines both. Finally we discuss implications of our approach for consistency and freshness of the user model information.

1 Introduction

Interpretation, reasoning, acquisition, and management of information about a user have long been the main topics of the user modeling community. These topics have usually been studied in the context of single application, often with a tightly coupled user model, represented in its own idiosyncratic way, often driven by the reasoning approach used in other parts of the system. This clearly has severe limitations for some key aspects of lifelong user modelling, or even more modest goals of user modelling that spans years or can support reuse by more than a single application.

There are several existing approaches that have the potential to inform design of an architecture that address these challenges. One move towards greater flexibility and reuse is the user model shell [14]. This strives to define a representation that has properties that are general enough for reuse by multiple applications. To make user models more readily available, there has been exploration of ways to create user model servers [9, 12]. Another trend has been the evolution of the service-oriented architecture (SOA), which aims to support applications in using information from loosely coupled services, each potentially using its own internal representations. Cloud computing [10] takes the broad SOA approach, giving people highly reliable and widely available access to services. These will play a role in the lifelong user model, at least as sources of the user's personal

information. This can inform the user model and may even constitute parts of it. For example, a service might maintain a medical user model, providing a repository for medical tests and other data as well as the user's health related model.

We propose to go beyond these approaches, defining *personal user model clouds*. Like cloud computing, we aims for high levels of reliability and availability, for use when and where the user's applications need to access it. However, our vision for the *personal clouds* is that each cloud defines just a subset of the user's complete model, where both of these may be distributed across multiple machines. As acknowledged in work on cloud computing, the issues of "privacy and confidentiality are ... perplexing" [10]: notably, while the user may want and need some of their user model reliably available to some applications, they may not want it available to others. So, we define the notion of personal user model clouds, both to capture the notions of availability and reliability from cloud computing, but distinguishing the different subsets of the user model as different clouds. This is an extension of our earlier notion of a *persona* [12] as a particular variant of a person's complete user model.

We illustrate the issues that motivate this work with the following scenario.

Alice wants to become a software engineer because she has had a long interest in computers. She studied the most challenging computing subjects available at high school and is now nearing completion of a degree in Software Engineering. Several programming subjects used a system call Reflect: this accepts assignment submissions, automatically grading them, provides code reading tasks and it also regularly asks her to assess her own knowledge of concepts. It has an open learner model, used for personalisation and to support reflection. Each subject's Reflect instance maintains its own learner model. This is integrated into the School learner modelling system and to the personal learner models of each student at their preferred machines. The same approach operates for other personalised teaching systems, such as LOGIC-ITA which teaches logic. Similarly, the School integrates information from the LMS, including subject examination results. The School regularly uses the learner models to review the success of its teaching and to identify areas for improvement to student learning.

Alice happens to meet Bob at the cafeteria and they realise they are doing the same capstone software project. They decide to discuss whether they should be in the same team. Alice uses her smart phone to review the parts of her learner model from the four years of her programming studies. She shares a summary view of this with Bob, to convince him that she could take the programming expert role in the group. He shares part of his user model to show he could be the algorithms expert. They decide to work together and send these user model views to other potential team members.

This scenario illustrates a long term user model that is derived from many sources. This is the type of user modelling that will characterise lifelong user modelling. Figure 1 shows a very high level view of the architectural issues in the scenario. Note that there are several sources of user modelling evidence: the teaching systems, both personalised and the conventional LMS and the exami-

nations. Each of the personalised systems has its own user model. There is reuse of the learner model by the School, as they mine the models of all learners from the multiple sources. Alice's full user model is delivered to her nominated (home) machine. A partial model is held on her phone. Other parts of her user model cloud for that context might be accessed via her phone, which connects back to her home machine. Importantly, Alice wants to be sure that the user model she shares with Bob is just the particular user model cloud that she intends. She defines this as her *sharable programming skill model cloud.*
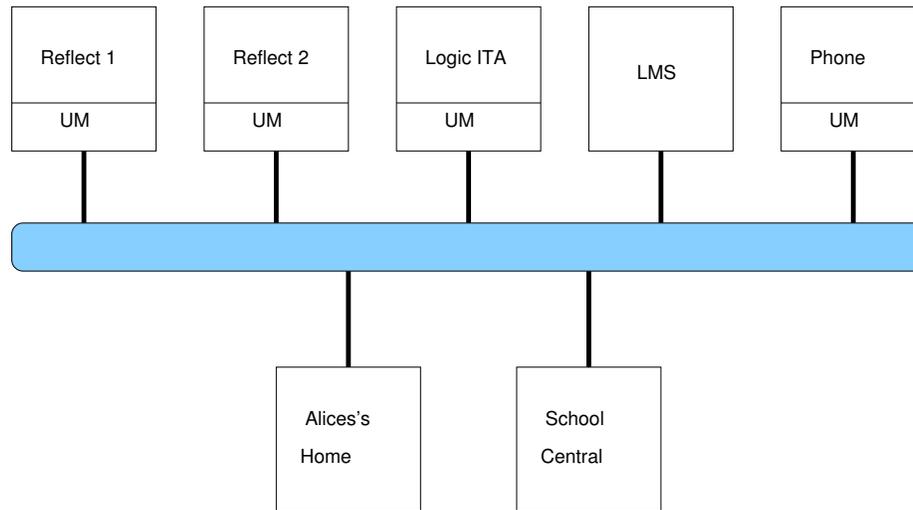

Fig. 1. Architectural overview.

Alice has other facets of her life beyond her formal studies. Each of these generates similar learner model clouds, for example, one for her health, another for her interests and yet another for her social life. Each of her personal machines may have various parts of each of these.

2 Data Model and Content of User Model

Key problems for user model exchange relate to the data model. This stores facts, potentially at different granularity levels from application to application. It may range from very specific models in one application, such as product preferences and user stereotypes for that context [2], to very generic ones, prescribing meta-primitives for exchange of user profiles as in UserML and GUMO – Generic User Modeling Ontology [11]. Sometimes, the generic semantic web data model is used to encode domain specific facts by instantiating concepts from accepted information models such as the ones in eLearning [8]. While some aspects of the

model can make use of an established ontology, applications may need purely local definitions, as we found in modelling learning in a course [13]. For example, in one of Alice's programming subjects, the lecturer may have defined the compulsory and optional parts of the course: these notions are important for modelling learning in the subject, especially for open learner models, but they are specific to that subject. In the long term, she can still make use of such specific models, for example, viewing it via an open learner modelling interface.

However, the lifelong model should be able to integrate modelling information across application, as in the case for the several programming subjects Alice studied. One approach, coming from semantic web research and inspired by data integration techniques, is to perform mappings between concepts and relations in data models at the conceptual model level or instance level. There are two main approaches:

- A common conceptual model approach — one conceptual model is taken as a standard and all the others are mapped to the global standard one
- A mapping to the neighbour — different conceptual models are mapped onto each other and the mappings between those which do not have direct mappings are derived through a transitive closure of mappings

There are several techniques to specify such mappings, including views, query languages, subclass relations and rule based specifications [15]. The main problem with these techniques is flexibility which is usually needed in user modelling systems and adaptability to different situations. The assumption behind these approaches is that the user model is neutral and the mapping can be neutral and used in different settings. However, these techniques can be used just over materialized facts and data. If the data supports different hypotheses, it is difficult to employ them without a reinterpretation or chaining of several reasoning components. This type of problem may arise in our scenario when the School wants to mine the full set of data, coming from the different subjects, each controlled by academics who defined or refine the models used in their courses.

Broadly speaking, it seems likely that ontology mappings may be feasible in many cases where they would be useful. For example, in the scenario, the School may establish some core parts of the learner model used in the tools already mentioned, as well as in curriculum specifications. It should be common that these can be linked to many core assessment elements used to create the learner model for each subject. If others are also defined, even if the central model cannot make use of them, there may be sufficient useful information to meet the goals of the data mining.

3 Ensuring that applications can send evidence to the user model and query it

This aspect requires that the architecture provides several facilities. It must enable an application to send relevant evidence to the user's model. In our scenario, this is needed to ensure that Alice's home model and the School central model

are updated. It must also support queries on the model: for example, when Alice wants to use her phone to acquire parts of her home model. For both of these, the architecture must enable an application to discover which machine has the relevant model and it must provide an API which the application can use to interact with the machine holding the model.

As we have noted, work on user model shells [14] and servers [9, 12] aims to make user model data and reasoning accessible for multiple applications. However, most of these have been closed, from the conceptual point of view: most support just one domain although they usually provide a query language, or API, to access it. The manipulation and exchange language ranges from knowledge based systems access languages, through database query languages, to semantic web query and reasoning languages. However, those languages and APIs differ between systems. This can pose a challenge for an arbitrary application to contribute to the models or to access required parts of them, distributed over servers and the user's own machines.

There has been some exploration of more flexible approaches to building user modelling servers. One of these is the Massive User Modeling System (MUMS) [4]. This uses a brokerage platform, data provider and modellers connected through a publish-subscribe mechanism. Such approaches may scale and could have a place in the lifelong user model.

A different approach, inspired by the demands of creating a framework for pervasive computing, is the PersonisAD Framework [7, 3], which makes use of either direct *tell* operations on the model, via a custom network protocol, as well as a publish-subscribe server to automatically collect evidence for the user model. It supports flexible *ask* requests on the model to resolve a value from the evidence when information, as needed. It supports distributed user models, with different parts of a person's model potentially residing on different machines. Another case is SEDONA [5]. Both PersonisAD and SEDONA support collection of user model evidence from multiple evidence sources. Notably, PersonisAD's active and distributed models operate in a way that can readily support distribution of the parts that constitute one of user model cloud. For example, when the user model in Reflect 1 is updated, the subscription rule on that component can send this part of the model to the user's home machine. This automated updating of the individual's personal master cloud is an essential part of our proposed architectural approach.

To deal with the problem of differing APIs, one approach is to create adapters which reside on solid vertical lines of Figure 1. This approach has been used in enterprise computing with service bus based systems or message oriented middleware. Each system connects to such an infrastructure through an adapter which transforms the internal data model representation to a common one for exchanging messages.

4   Ensuring Consistency

In databases, there is typically a goal for consistency, so that any application's query returns the same, current, fresh result. In our vision, this is neither possible, nor desirable, in general.

Some of the reasons that it is impossible are those from standard database consistency, for distributed databases. One that is significant relates to cases where some of the machines involved are not available at the time of either an update or of a query. For carried devices, such as phones, they may be turned off to conserve battery. For devices, such as Alice's home machine in the scenario, the machine may also be turned off or otherwise inaccessible. The Personis representations cannot solve this. However, as all evidence is timestamped with the time it was added to the model, this provides an indication of how current a resolved value is. It would also be possible to use it to resolve different values coming from different servers.

Another case arises where multiple servers are concurrently collecting evidence about the user. While this problem is not of the same order as large data bases with large numbers of writes, it may arise in the case of concurrent collection of evidence from programs the user interacts with and pervasive sensors. There has been some work to explore ways to ensure a consistent user profile view among concurrently running collaborating adaptive systems and a set of associated user modelling servers. For example, one approach involved each server notifying all currently active adaptive systems about any updates made to the user model being used by them [1]. This requires that adaptive applications register with the user model servers when they start and indicate when they end. Then the server can maintain a list of applications that need to be informed of changes. This can be achieved by subscribing/unsubscribing from the notification service provided by the user model servers. Similarly, there was a synchronisation between the servers, ensuring that they can support consistency integration. This is based on the registration and propagation of identification and context of the adaptive tasks during the concurrent access and manipulation of the user model data. Other systems, including SEDONA and PersonisAM address these problems by a simpler locking mechanism during writes and ignoring the need for integration across multiple servers.

The choice of an appropriate and effective consistency and freshness mechanism will depend upon the nature of the user models. For example, it may be argued that some known level of potential inconsistency or staleness might be acceptable, especially if there is a modest rate of writes onto the models. It may also be acceptable even when there are higher write-rates but the precise value is not critical. Indeed, if there are only parts of the model where a reliably consistent and fresh value is important, then it may acceptable to have a more relaxed management mechanism for the rest of the model.

The issue of whether consistency is *desirable* is more subtle. It interacts with our notion of a set of personal user model clouds because we envisage that a query to one cloud would frequently return a different value from that of another cloud. Suppose, for example, that Alice defines several clouds, one each for her

university studies, the subset of this associated with her programming studies, her sport and fitness activities, her medical and health model, and her other personal model. Different people, and applications associated with them, would be allowed access to the relevant subset of these. So, for example, her sports coach may be allowed access to her sport and fitness model and, when she meets her coach, Alice could retrieve this on her mobile phone to share and discuss. When she met Bob, in the scenario, she shared the programming studies model and this is what she sent on to the other potential team members. The differences go beyond subsetting the components within the model. It can also involve the choice of resolver used to conclude the value within the model. For example, in reporting a user's location, MyPlace used various resolvers to control privacy [6]. A similar level of control might be applicable for many other parts of a long term user model, for example parts associated with the user's education and health.

Our architectural proposal has two main mechanisms for transfer of user model information. One is directly from applications. The other mechanism is needed for synchronising models, such as the Reflect teaching system, to Alice's home machine. For both, we propose that most cases can be adequately handled with existing database locking for writes. This must be supplemented by a mechanism to deal with the potential for disconnected operation. For this, we propose a message-based asynchronous mechanism. For reads, we propose that the architecture provides information about the freshness of the result, leaving the application to determine a suitable course of action. For example, this might involved informing the user. Equally, it might involve a wait and re-try cycle. In some cases, the application may choose not to perform a personalisation action. Before considering an elaborate mechanism for this case, we will evaluate the simpler approaches and gain understanding of its limitations.

5 Conclusions

We have explored a range of issues that are drivers for the design of an architecture for collections of personal lifelong user model clouds. These clouds aim to ensure that the user's applications can reliably access the relevant user model cloud for them, within the constraints of the connection of relevant machines. Such constraints are partly due to the demands of privacy, with the options for the user to be aware of which machines hold certain parts of their complete user model. So, for example, particularly private information may be restricted to a particular home computer, even though this means that this information may not always be available. A key part of the motivation for user model clouds is to support user control of their model and its availability to different applications. We have glossed over the challenging task of creating an interface that will enable users to manage this. Our proposed architecture is lightweight, with the potential to support collections of user model servers and repositories, with the complete model being conceptually segmented into the particular clouds intended for different uses, applications and people.

Acknowledgments

References

1. M. Alrifai, P. Dolog, and W. Nejdl. Learner profile management for collaborating adaptive eLearning applications. In *Procs of the Joint Intern'l Workshop on Adaptivity, Personalization & the Semantic Web*, pages 31–34. ACM, NY, 2006.
2. Liliana Ardissono and Anna Goy. Tailoring the interaction with users in web stores. *User Model. User-Adapt. Interact.*, 10(4):251–303, 2000.
3. M Assad, D J Carmichael, J Kay, and B Kummerfeld. PersonisAD: Distributed, active, scrutable model framework for context-aware services. In *Procs of Pervasive 07, 5th Interna'l Conf on Pervasive Computing*, volume 4480 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2007.
4. C Brooks, M Winter, J Greer, and G McCalla. The massive user modelling system. In *ITS'2004: Intelligent Tutoring Systems*, LNCS, Maceio, Brazil, 2004. Springer.
5. P Brusilovsky, S Sosnovsky, and M Yudelson. Ontology-based framework for user model interoperability in distributed learning environments. In G. Richards, editor, *World Conference on E-Learning, E-Learn 2005*, pages 2851–2855, Vancouver, Canada, 2005. AACE.
6. D Carmichael. *Myplace: supporting scrutability and user control in location modelling*. PhD thesis, 2008.
7. D. J. Carmichael, J. Kay, and R. J. Kummerfeld. Consistent modelling of users, devices and sensors in a ubiquitous computing environment. *User Modeling and User-Adapted Interaction*, 15(3-4):197–234, 2005.
8. P Dolog and M Schäfer. A framework for browsing, manipulating and maintaining interoperable learner profiles. In Liliana Ardissono, Paul Brna, and Antonija Mitrović, editors, *Proc. User Modeling 2005*, volume 3538 of *LNAI*, pages 397–401, Edinburgh, Scotland, UK, 2005. Springer.
9. J. Fink and A. Kobsa. A review and analysis of commercial user modeling servers for personalization on the world wide web. *User Modeling and User-Adapted Interaction*, 10(2):209–249, 2000.
10. B Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.
11. D Heckmann and A Krueger. A user modelling markup language (userml) for ubiquitous computing. In P Brusilovsky, A T Corbett, and F de Rosis, editors, *In Proc. of User Modeling 2003*, pages 393–397, Johnstown, PA, 2003. Springer.
12. J. Kay, B. Kummerfeld, and P. Lauder. Personis: a server for user models. In *Proceedings of AH 2002*, volume 2347 of *Lecture Notes in Computer Science*, pages 203–212. Springer-Verlag (Berlin, Heidelberg), 2002.
13. Judy Kay and Andrew Lum. Exploiting readily available web data for reflective student models. In *Proceedings of AIED 2005, Artificial Intelligence in Education*, pages 338–345, Amsterdam, The Netherlands, 2005. IOS Press.
14. A Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(49):49–63, 2001.
15. K van der Sluijs and G-J Houben. A generic component for exchanging user models between web -based systems. *Int. J. Cont. Engineering Education and Lifelong Learning*, 16(1–2):64–76, 2006.