User Models Sharing and Reusability: A Component-based Approach

Eyal Dim and Tsvi Kuflik

The University of Haifa, Mount Carmel, Haifa 31905, Israel
dimeyal@bezeqint.net, tsvikak@is.haifa.ac.il

Abstract. The current state of affairs in user modeling is that user models are developed ad-hoc, as part of a specific application. The proprietary user models are evidence of the lack of standard user-modeling processes and the amount of unnecessary rework done. Nowadays, when people tend to share information, open source is available, and Component-based software development is a common practice, it is time to adopt it for user modeling as well. The Component-based user-modeling approach suggests a practical alternative to the ad-hoc development of user models: user-modeling components that can evolve and be developed collaboratively and incrementally by the user modeling community, enabling reuse and flexibility, leading towards new and advanced user models built from existing components.

Keywords: user modeling, component-based user-model, reusable user models, ubiquitous user modeling.

1 Introduction

Current systems that provide personalized services to their users develop their own proprietary User Models (UMs) in an ad-hoc manner. This is due to the following main reasons: (i) System's developers focus on specific characteristics of their users in order to deliver a specific service (e.g. a movie recommendation). (ii) The user modeling community does not have commonly accepted standards for UMs and their structure. (iii) Reusable user modeling components are not publicly available. Finally, (iv) the amount of effort required to develop a standard UM that holds a comprehensive set of attributes is too big to be carried out in a single application. Over the years, the user modeling research moved from suggesting complete, monolithic solutions in the form of user modeling servers [Kobsa, 2007] and user-modeling ontological representations [Heckmann, 2005], to partial and dynamic solutions in the form of mediation and other aspects of interoperability and interlinking, while exploiting information that may be available in various online sources (like social networks) [Carmagnola et al., 2011; Leonardi et al. 2010]. We propose a different approach that may support collaboration in developing UMs. This goal may be achieved by dividing the UM into small, standard and reusable multipurpose building blocks that may be integrated into more abstract UMs as needed. The proposed approach, a Component-based approach for user modeling, is based on the Component-based development approach, where a component is "a reusable unit of deployment and composition" [Crnkovic et al., 2002]. This is an incremental approach for building UMs by using components that will follow a standard form and will be publicly available to UM developers. This will encourage developers to adopt standard component definitions and create and use inventories of components in the cloud, while improving the components over time and enabling their sharing and reuse, as well as exchanging user modeling data across applications. For example, a Component-based UM may use vital signs evidence, handled by a heartbeat component and a blood pressure component to infer the user's health-state for serving a medical application, while another UM may use the same components to infer the user's excitement-level for a shopping application (see fig. 1 and more detailed explanation below). In both cases, the inference would be based on the same vital signs data, if standardized and ready for reuse. Furthermore, the excitement-level attribute (as well as the health-state attribute) may become a component too, available for the development of UMs that need this component to provide other services.

The proposed approach suggests that a user-modeling component, will represent a **single attribute**. Such a component is small, bounded, and easier to design in comparison to the entire UM. Moreover, the component is **standalone**, and may reside anywhere in the cloud. It may use any inference, data fusion, reasoning, or learning method. A reusable component in this case may become a building block in any UM. In the future, such components would be available and would evolve to become a standard de facto. Dividing the UMs into simpler and smaller components may enable the recruiting of multinational effort to map and generate these components. The components, their dynamically expanding inventory and relevant methodologies could become a user-modeling standard supporting component suppliers, UM designers and service applications developers.

2 Background and Related Work

Component-based development is a software engineering methodology, which supports the development of rich software applications by dividing them into independent components as suggested by McIlroy [1968]. A component, according to Estublier and Favre [2002] is an abstract box that encapsulates its properties and processing. The Component-based development approach focuses on sharing and reuse of components that has a high potential for improving the efficiency of system design and development [de Almeida et al., 2005]. Stojanovic et al. [2006] presented a generalization of component design that may hold information, be sensitive to events and context, and supply services, which are constrained by conditions. Estublier and Favre [2002] and Szyperski [2002] discussed various Component-based development frameworks that matured for commercial usage, such as: JavaBeans, COBRA, CORBA, COM, DCOM, COM+, MTS, CCM, .NET, and more. As part of the context-awareness research, Dey [2000] presented the concept of the widget. The widget is a component responsible for acquiring a certain type of context information and making it available to applications in a generic manner, regardless of how it is actually sensed. Baldauf et.al [2007], presented some other tools that enable distributed parallel access to data while encapsulating low-level data. Hofer et al. [2002] presented an object-oriented approach, where specific objects supply specific lower level interpretation of sensor information such as location, time, and network to a context server that supplies relevant information to applications.

As for user modeling, the proposed Component-based UM approach requires both the selection of those characteristics and features that can be downsized or attached to a single-attribute component and the utilities that handle the inventory of user modeling standalone components. Nowadays, UMs are still developed ad-hoc, as part of a specific application, requiring mediation in order to allow reuse [Berkovsky et al., 2008]. As for a UM breakdown, the General User Model Ontology (GUMO) and UbisWorld ontology [Heckmann, 2005] presented the UM as hierarchy and breakdown of concepts and attributes, implemented by using the User Modeling Language (UserML). The contribution of GUMO and UbisWorld was in the methodological representation of detailed user-modeling ontology, including concepts and their hierarchies. However, while GUMO and UbisWorld were important attempts towards standardization and coverage of all aspects of ubiquitous user modeling, this was their pitfall. It is difficult to use them due to their comprehensive nature on one hand, and the concepts they do not include on the other hand. Grapple and GUMF [Leonardi et al., 2010] are based on the concepts suggested by GUMO for attribute definition, metadata, having the benefit of rule-based reasoning, as well as web content retrieval and knowledge buildup. While they suggest both an attribute definition and a UM framework, the attributes are kept within the system, and are not standalone, and the reasoning process is limited to rule-based reasoning.

3 Suggested UM Structure

A reusable UM component should be small enough while containing meaningful information useful for a variety of UMs. The inventory of such components in the cloud should be rich enough to represent the concepts within a UM ontology. Each component should be able to gather data, use an inference process to generate new information, store it, and be able to provide the information to services or other components. Since we are interested in a standalone single-attribute component, its structure needs to be standardized, and support utilities such as explanation and privacy. Such components may collect evidence from sensors (the same way widgets are doing it), furthermore, they may store new information inferred from the evidence and provide it to services as needed. Their output may serve as input to new components, generat-

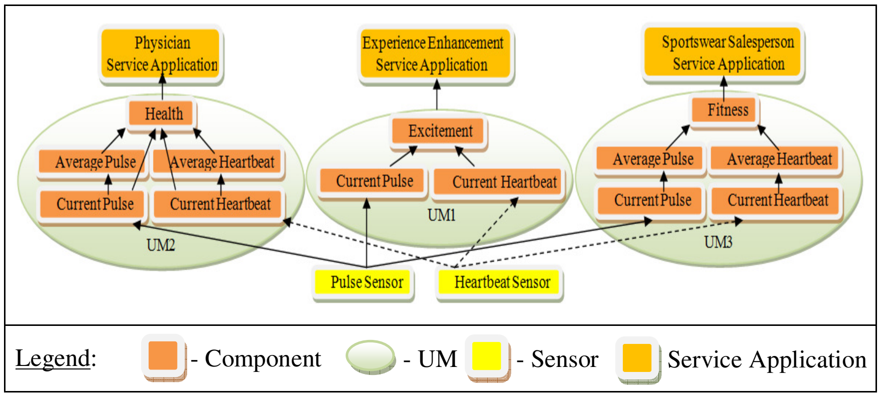

Fig. 1. Multiple UMs sharing the same component in different compositions

ing new concepts through composition of standalone components. For example (see Figure 1), a sportswear salesperson, trying to fit a product to a customer based on the user's fitness, may activate a UM service that returns the fitness level based on the user's average heartbeat and pulse. Another UM may serve a physician by assessing the patient's health based on the comparison of the patient's current pulse and heartbeat with their averages supplied by reuse of these components. A third UM may reuse the current pulse and heartbeat to assess the current user's excitement level. All three UM's in this example share the same sensor data and reuse some components, but differ in the component compositions and the higher-level inference: the components "health", "excitement" and "fitness".

3.1 The UM Basic component

Each *basic component* handles a single UM attribute. It has six parts (see Figure 2): *input port/s, output port/s, attribute generator, attribute repository, attribute access*, and *meta-data & tools*.

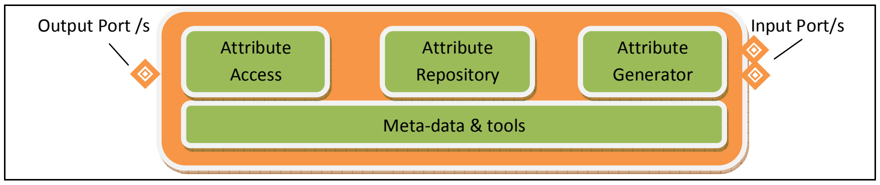

Fig. 2. The Basic Component

The *attribute generator* reads the input data from the *input port/s* and manipulates it. It may simply save the data in the *attribute repository*, or use more complex processing of the input data, such as: data fusion, inference, and automatic learning. If there is no data input, the *attribute generator* may use default data to initiate the attribute value. Once the data manipulation is complete, the attribute data is ready to be time tagged, and stored in the *attribute repository* with value-related characteristics such as accuracy, confidence level and expiration date [Heckmann, 2005]. The data may be a single value, a vector of temporal accumulation of values, or any other data structure, as long as it represents a single attribute. When a service application or another component needs the attribute's data, it may access the basic component through the *output port/s* that return the required value in a predefined format. The *output ports* are connected to the *attribute access* that verifies whether the service (or another component) is authorized to access the data, based on considerations such as privacy. Finally, there is the *meta-data & tools* part. It adds information that relates to the component itself but not to the value currently stored within the component, such as: UM instance identifier (e.g., the actual user or group identifier) [Heckmann, 2005]; or a definition of the attribute (help function), and explanations/scrutability [Hechmann, 2005; Kay and Kummerfeld, 2006].

3.2 The broader view

Once developed, components may be used by a UM designer as needed for any specific UM. Over time, a large inventory of components may be developed and ideally stored in the cloud. In most cases, only a subset of that inventory may be required for modeling a specific domain. Each such subset may be an independent UM assembled from basic components, available as part of the entire inventory. The component's metadata will enable the UM designer to find and select the required components (using keywords or component description), and to understand the component's behavior. Figure 3(a) illustrates UM components stored in the cloud, while Figure 3(b) shows how UMs are built by selecting individual components from for specific applications during design. The components are reusable, therefore a component may be used by more than one UM (see the intersection of UM1 and UM2). Finally, figure 3(c) presents the instantiation of one of the models (UM1) for three specific users (John, Marry and Jane) at run-time.

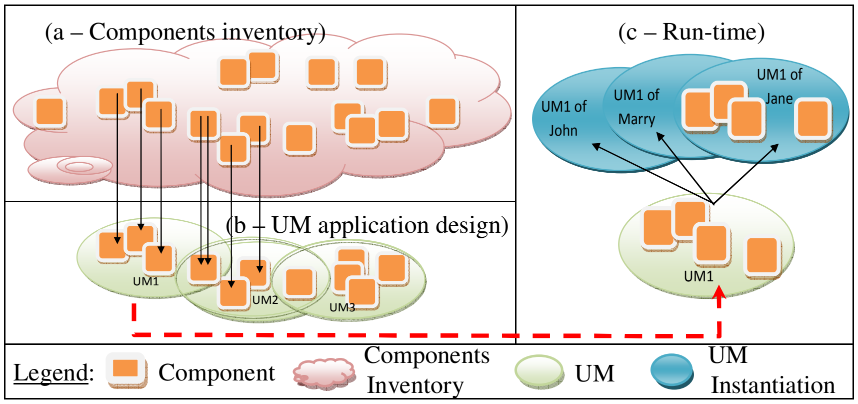

Fig. 3. Components, components inventory and UMs

4 Conclusions and Future Work

This paper presents a concept and a vision of a Component-based UM approach. Single attribute standalone components and their compositions may enable building abstract UMs. Each UM will be composed of the components that fit its purpose. Acceptance of this concept can enable components variability, reusability and flexibility. Multiple vendors and freeware producers may compete over the future market of UM components. Standards would evolve for specific components. Aggregation of components or component-compositions would enable simple as well as complex services. Experts would supply components in their field of expertise. New methods and tools would enable components management, publication, and accessibility.

There are challenges that future research should address, such as: demonstration of how the proposed approach can support the current user modeling methods (such as feature-based filtering and collaborative filtering); standardization of the component structure; maintainability; managing consistency; UM mediation through detection and sharing of standard components across UMs; reliability and validation of components; managing components' errors and fault tolerance; components plug-and-play techniques; and considerations for scalability, efficiency and performance.

The component-based user modeling approach may set the necessary conditions for sharing and reusability of UM structures as well as UM content. The dynamically expanding inventory of components and supporting methodologies may become UM standards de-facto. This process may encourage multinational collaboration serving the benefit of users worldwide.

References

1. Baldauf, M., Dustdar, S. and Rosenberg, F.: A survey on Context-aware System. In: Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, pp.263–277, (2007).
2. Berkovsky, S., Kuflik, T., Ricci, F.: Mediation of User Models for Enhanced Personalization in Recommender Systems. In: User Model, User-Adapt. Interact. 18(3), pp 245–286, (2008).
3. Carmagnola, F., Cena, F., Gena, C.: User model interoperability: a survey. In: User Model User-Adap Inter 21:285–331, (2011).
4. Crnkovic, I., Hnich, B., Jonsson, T., and Kiziltan, Z.: Basic Concepts in CBSE. In: Crnkovic, I. and Larsson, M. (edts.): Building Reliable Component-based Software Systems, (2002).
5. De Almeida, E.S., Alvaro, A., Lucredio, D., Garcia, V.C.; , de Lemos Meira, S.R. : A survey on software reuse processes. In: IRI -2005 IEEE International Conference on Information Reuse and Integration, pp 66-71, (2005).
6. Dey, A. K., Abowd G. D. Towards a better understanding of context and context-awareness. In: CHI'2000 Workshop on the What, Who, Where, When, and How of Context-Awareness, (2000).
7. Estublier, J., and Favre, J. M.: In: Crnkovic, I., and Larsson, M.: Building Reliable Component-based Software Systems, Artech House, Inc., pp. 57-86, (2002).
8. Heckmann, D.: Ubiquitous User Modeling. Akademische Verlagsgesellschaft Aka GmBH, Berlin, (2005).
9. Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G.and Altmann, J.: Context-awareness on Mobile Devices – the Hydrogen Approach. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, (HICSS'03), IEEE, pp.292–302, (2002).
10. Kay J. and Kummerfeld B.: Scrutability, User Control and Privacy for Distributed Personalization. In: Proceedings of the CHI2006 Workshop on Privacy-Enhanced Personalization, pp 21-22, (2006).
11. Kobsa, A.: Generic User Modeling Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.): The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science, LNCS 4321, Springer-Verlag, Berlin eidelberg New York, pp 135-154 (2007)
12. Lnardi E., Abel F., HeckmannD., Herder E., Hidders J., Houben G. J.: A Flexible Rule-Based Method for Interlinking, Integrating, and Enriching User Data. In: Proc. ICWE 2010,(2010).

13. McIlroy, M.D.: Software Engineering: Report on a conference sponsored by the NATO Science Committee. In: NATO Software Engineering Conference, NATO Scientific Affairs Division pp. 138-155 (1968)
14. Stojanovic, Z., Dahanayake, A., Sol, H.: An Approach to Component-based and Service-oriented System Architecture design. In: De Cezare, S., Lycett, M., Macredie, D. R. (edts.): Development of Component-based Information Systems, M. E. Sharp, Inc., pp 23-48, (2006).
15. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Professional, Boston, (2002).