Towards a Generic User Model Component

Kees van der Sluijs, Geert-Jan Houben

Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{k.a.m.sluijs, g.j.houben}@tue.nl

Abstract. The increasing need for personalization forces developers to automatically adapt their applications to individual users. In order to realize this, an application needs a model of the user with as much and as accurate data as possible. However, users typically divide their time over many applications that individually are limited in their user modeling and therefore can gain from joining forces. This joining of forces boils down to establishing semantic interoperability of user models. Semantic interoperability already proved to be extremely hard in the past, but the emerging Semantic Web might offer just the mechanisms that we need. From our ongoing research this paper presents the Generic User Model Component (GUC), a generic component that provides user model storage facilities for applications. Moreover, by utilizing Semantic Web technology (e.g. RDF(S), OWL) it also supports the exchange of user data between applications. GUC is one of a series of generic components for configuring a complete Adaptive Web Information System. We present a high-level architecture for GUC, realizing different levels of user model server functionality. Moreover, we discuss how GUC can function in certain scenarios by adapting its configuration.

1 Introduction

The Web has a very large and heterogeneous audience. At the same time, the amount of Web applications grows constantly as more and more organizations replace traditional user-communication with communication through the Internet. These developments make an increasing amount of Web developers realize that the traditional one-size-fits-all approach no longer works. They want to personalize their application to the user.

In order to personalize the communication with the user, applications need to maintain some kind of model of the user. This model should describe properties of the user, such as preferences, behavior, knowledge and other relevant facts. In order to optimize the personalization, this model should be as accurate and as complete as possible. However, users divide their time over many applications on many devices. They are typically not willing to invest a lot of time to explicitly fill this model for every single application they use. The result is that data on the user is fragmented over many applications. The amount of information of a user in a user model of a single application is usually rather limited. Furthermore, many applications share data on the user, which means that users often have to input data that already was provided for another application. Therefore, it would be enormously advantageous if we have a way for applications to share their knowledge on the user.

Sharing of knowledge on the user between applications requires semantic interoperability for their user models. This is in general extremely difficult and requires a very high degree of alignment between the applications on syntax, structure and semantics. The past has taught us that in a heterogeneous environment such as the Web it is near to impossible to enforce applications to use a prescribed syntax, structure and semantics.

However, the recently emerged Semantic Web (SW) offers mechanisms that might help us in this quest. The SW provides a common framework for structuring data in a syntax-independent way. It allows the flexible definition of data structures and offers mechanisms to define relations between those structures. Moreover, SW's well-defined semantics allows to reason on its data. These ingredients fulfill the basic requirements to establish semantic interoperability.

Earlier work on user model servers and semantic interoperability focussed primarily on the syntactic problems. The basic problems were already identified in the work of Orwant called 'Doppelgänger' [1]. This work focussed primarily on the gathering and distribution of data and the use of several learning techniques. More recent approaches like UserML [2] and GUMO [3] focus on creating a common language and ontology for communication of user models. Another example is MUMS [4] in which communication interoperability is established by a generic communication framework and semantic interoperability is stimulated by offering an ontology database for the sake of reuse of (parts of) ontologies.

In this paper we will describe the Generic User Model Component (GUC). GUC is a generic component that utilizes Semantic Web technology to provide user model server capabilities. Requirements for such a system are *generality, expressiveness and strong inferential capabilities* [5]. We developed GUC such that it satisfies these requirements. In Section 2 we introduce GUC and provide some context for this component. In Section 3 we describe GUC's architecture in detail. Furthermore, we will present different configurations with which GUC can function within a number of scenarios in Section 4 and discuss some open issues in Section 5.

2 Motiviation

GUC is one component in a series of generic components for model-based application development. They are developed to offer designers components that after configuration can be linked together into a complete component-based Adaptive Web Information System (AWIS). This component-based architecture is based on Hera [6] and AMACONT [7] technology. Formerly, we created a generic transformation component GAC [8], specially for different transcoding steps, a Self-Adaptive GAC component SAG [9], that allows dynamic self-configuration, and HPG, a component for hypermedia presentation generation. Forthcoming is a generic component for dealing with heterogeneous data sources.

GUC is a generic component that offers functionality to store data models for applications and to exchange user data between those models. GUC can be used as part of an AWIS, but can also be used as a stand-alone user model server. In Section 3 we will explain the GUC architecture.

3 GUC

Within our architecture we use the terms *UM-based applications* and *users*. With *applications* we mean all entities outside the GUC that want to utilize the GUC to store data on users. This might be applications in the classical sense, but also sensors, agents and other processes. With *users* we mean all application users for which the applications want to store a model. We focus on the user as a human individual, but this might as well be extended to groups, and to other applications (e.g. agents) that access the applications.

3.1 A User Model Repository

Applications can "subscribe" to GUC. Subscribed applications can then request and upload user data of particular users. In order to provide this functionality for an application, GUC requires a *schema* that describes the data structure of the user model for that application. GUC stores these application schemas in its *application schema repository.* If the application schema is present in GUC, the application can upload instances of that schema for particular users, i.e. for every user that accesses an application an instance of this schema is stored in GUC. An application may request for a particular user the corresponding instance of its application schema. We call such an instance an *user application-view*(UAV). On such a request, GUC will simply send the UAV as the application last uploaded it. Figure 1 shows the basic GUC functionality of user model repository.

3.2 Mapping schemas

As we explained earlier, we do not use GUC solely for the storage of user models, but also to exchange data on the user between different applications. If we look at our user model repository, it means we want to exchange data between UAVs. Suppose we want to exploit data from UAV a within UAV b. To do that, we first transform the data of UAV a into the structure of UAV b. Second, this transformed UAV a' has to be integrated into the existing UAV b. This transformation and subsequent integration form what we call an *instance mapping.*

The instance mapping is generated from a *schema mapping.* A schema mapping from a schema A to a schema B contains a specification of how all constructs in schema A are mapped to corresponding constructs in schema B. Such a schema mapping contains the rule types as shown in Table 1. Note that the elements in the table like *name, person, age*, etc. are classes. For the semantical mapping between birth date and age an interpretation directive has to be given, in this case a formula that indicates how age can be calculated by subtracting the

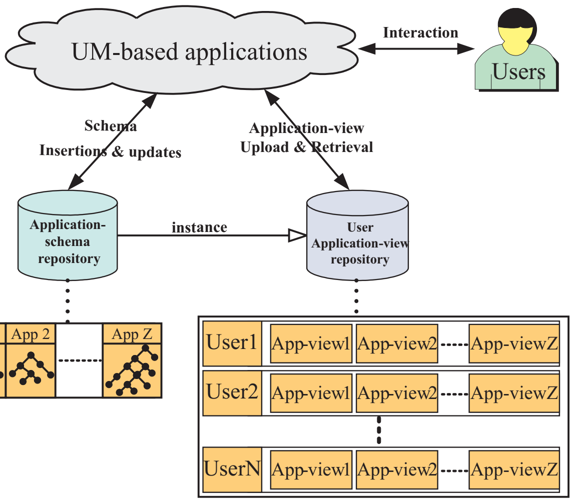

Fig. 1. Basic GUC architecture

value for the birth date from the current system time. The data reconciliation rule type helps to define what to do if a value in the transformed UAV already exists in the UAV that it should be integrated in: it is possible that the value is concatenated with the current value, or that the current value is replaced, or that a decision is made based on a given formula. Examples of such decisions are decisions based on time-stamping (e.g. latest time-stamp wins) or based on a trust value (of the user) for the corresponding applications.

Rule type	**Example**
Syntactical transformation	$Name \Rightarrow PersonalName$
Structural transformation	$[person \overset{hasData}{\rightarrow} \{\} \overset{hasName}{\rightarrow} Value] \Rightarrow$ $[person \overset{hasName}{\rightarrow} Value]$
Semantical transformation	$BirthDate \Rightarrow Age$ {Age = Systemtimestamp - Birthdate}
Data reconciliation	e.g. concatenate, replace, decide based on formula

Table 1. Schema Mapping type of rules

We designed our GUC such that the schema mapping is interpretable by a generic piece of code inside the GUC to generate the instance mapping: given a schema mapping the GUC can generate the mapping at instance level. For this, we must be able to express schemas and their instances, and furthermore we must be able to syntactically, structurally and semantically relate these schemas and their instances. This is where we can effectively use SW technology. The SW initiative provides us with a generic way to structure data. Furthermore, the SW language OWL [10] provides just the mechanisms we need to relate these data structures. We can use OWL constructs like equivalent, sameAs, differentFrom, AllDifferent and distinctMembers that allow expressing relationships between schemes, and we can use this to apply the mappings to concrete UAVs.

Where do the required schema mappings come from in the GUC? These mappings are delivered by our mapping module. For this, the mapping module requires the source schema, say A, and the target schema, say B. As the mapping between schema A and schema B only has to be constructed once, it can be created by the (human) designer. Since it can be a laborious task, we present the designer with some supporting techniques.

The designer can use matching and merging techniques to match two input schemas and create a merged schema that is a union of both input schemes. There exist numerous heuristic algorithms applicable to an SW language like OWL that can generate such a merged schema. These algorithms exploit heuristics such as name similarity, thesauri, schema structures, value distribution of instances, past mappings, constraints, cluster analysis, and similarities to standard schemas. Our GUC architecture allows to use such matching algorithms, but whether such an algorithm is used, and if so which one, is optional. Examples of such algorithms are similarity flooding [11] and the machine learning based matching software GLUE [12]. For a survey on these techniques see [13]. Whatever the algorithm used for schema matching, the result must be verified and possibly be edited by hand before it can be effectively used. This is because semantical structures may not be interchangeable just like that. Schema elements that seem the same may not be interchangeable on instance level. Consider for instance the related concepts user-name and password. While the semantical meaning of these concepts might be completely the same for two applications, the concrete values for these concepts for a particular end-user still might not be interchangeable for those applications. The syntactical and structural transformation mappings can easily be automatically generated by the mapping generated in the matching process. The semantical transformation rules and the data reconciliation rules are however typically added manually. The GUC can support this process by offering templates for often occurring rules.

With the matching and merging techniques, we can construct a *combined ontology* of the application schemas. For instance, consider the mapping module merges schema A and schema B into schema M (or $A \cup B$), via a combined ontology. This results in the (partial) mappings from A to M and from M to B, but also from B to M and M to A. In this way, we can map schema A to schema B, via schema M, but also map schema B to schema A. Thus, we are able to easily "attach" other schemas (e.g. schema C) to M as well. In this way we can attach a new application-schema to GUC by simply merging it with the combined ontology and define a mapping to and from the new application-schema and the combined ontology.

Figure 2 depicts the schema mapping module as we described it in this section.

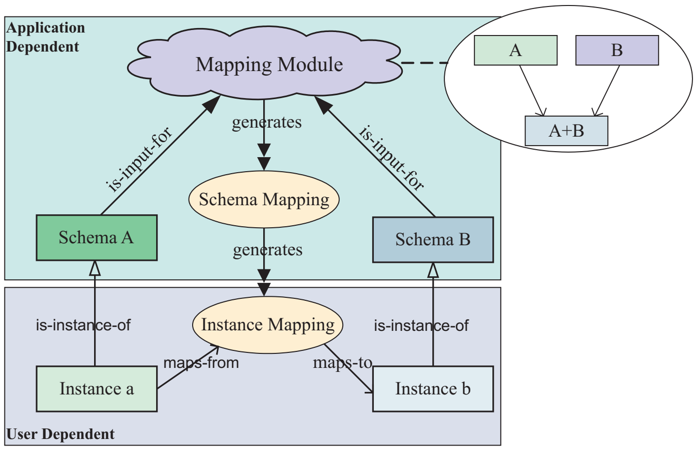

Fig. 2. Schema Mapping Module

3.3 Complete Architecture

We combine the basic architecture of a user model repository (Figure 1) together with the schema mapping module (Figure 2) to a user model server architecture for GUC. In our architecture, we make an explicit distinction between the parts that are application-dependent and user-dependent.

For every application we maintain an application schema and we generate a schema mapping of that schema to the combined ontology. For every user a *UAV repository* is maintained. In this repository an UAV is stored for every application that the user uses. All the UAVs are combined in the *GUC global user model*, which is a (partial) instance of the combined ontology. This structure contains all the data that is known of the user. The global user model is created by the *GUC data manager* by applying the mappings that are stored in the *schema mapping repository* to all UAVs in the UAV repository.

When an application requests data on a particular user for the first time, GUC will create an UAV for the user that is an instance of the corresponding application schema in the application schema repository. The GUC data manager will try to fill this UAV on the basis of data that is stored in the global user model. To this end, it can again apply the mappings in the schema mapping repository. The result will be a partially filled user model. With this partially filled user model the application can choose to limit the user-interaction to request only the information still unknown. The combination of all these elements results in the complete architecture of GUC as depicted in Figure 3.

Currently we are progressing with the implementation of GUC. We implement GUC as a Java servlet. We use the SW languages RDF(S) and OWL. For the repositories we use the open source RDF database Sesame [14]. GUC can with little configuration be adapted to applications that output their user profiles in a SW-format. For the user model repository functionality, only once schema

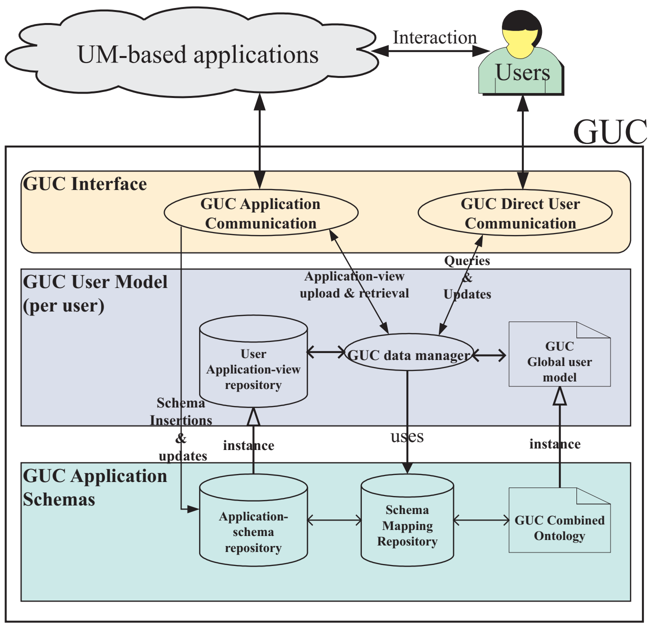
UM-based applications
Interaction
Users
GUC
GUC Interface
GUC Application Communication
GUC Direct User Communication
GUC User Model (per user)
Application-view upload & retrieval
Queries & Updates
User Application-view repository
GUC data manager
GUC Global user model
Schema Insertions & updates
instance
uses
instance
GUC Application Schemas
Application-schema repository
Schema Mapping Repository
GUC Combined Ontology

Fig. 3. The complete GUC architecture

of the data has to be provided to GUC to let cooperate with the application. For the full user model server functionality a mapping has to be generated by the developer, manually or supported by matching algorithms and templates.

4 Configuration

GUC is a versatile component in the sense that it can be used within many scenarios regarding user-application interaction with only little configuration. The architecture still leaves a lot of freedom for different applications of GUC that might be beneficial given a known context scenario. In this section we provide configurations that can be beneficial for a number of scenarios.

4.1 Front-end

By configuring GUC as a front-end application we basically deploy GUC as a personal help for the user. Even though several options exist we favor the arrangement as depicted in Figure 4, namely to use GUC as a front-end application. In this way users would communicate with the applications *through* GUC. This is particularly useful, as in the future we plan to use GUC as part of a suite of generic components that together form an AWIS interface for applications. We then should offer the user a simple but powerful control mechanism to maintain the user model and put the user in control on what data gets communicated to the applications. This can potentially boost the user's trust in the system. This also provides a basis to attach *learning mechanisms* to GUC to learn the user's behavior.

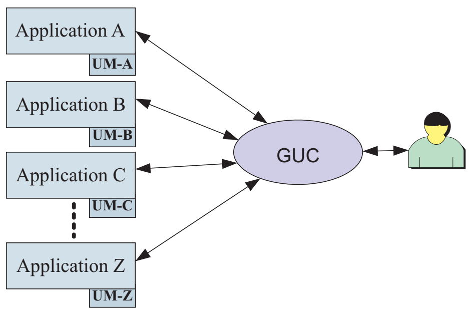

Fig. 4. GUC as front-end application

4.2 Legacy Applications

Besides applications that actively subscribe to GUC we might want to also make GUC backwards compatible, as there already a lot of applications maintaining user models. By using GUC as a front-end for users we not only have the possibility to maintain the output user models of the applications, but also to maintain models of the data inputed by the user to applications.

We also have to consider that many "legacy applications" do not use SW-technology (yet). As the SW languages allow to express virtually any data structure we opt for applying wrapping techniques to support applications that do not natively support RDF(S) and OWL. We regard the actual wrapper, for wrapping a given data structure into a Semantic Web data structure, as a different component that we will develop separately. The modular design of our component collection would however easily permit to couple such a component to GUC.

4.3 Mobile devices

By extending the configurations of the former sections we can create a *personal assistant*. This would be in line with our research in the IST-project MobiLife [15]. As mobile devices get more powerful and ubiquitous an option is to let users "carry" their profile with them on such a device. As more and more organizations offer Web alternatives for filling in forms, GUC can be able to help filling in these forms. Many of these applications will however not be familiar with GUC or will not use SW technology.

By extending the scenario's of the Section 4.1 and Section 4.2, we can deploy GUC on a mobile device and utilize it for both Web applications and "local" applications. With local applications we mean applications that are accessible via the local network of the mobile device. The user can thus use applications on multiple devices in a personalized way as GUC can send parts of the user model when needed.

5 Discussion

A few questions are yet unanswered. For instance, what about the scalability of GUC? We claim that this will not be a major problem as we plan to go towards the mobile scenario discussed in section 4.3. The most computation-intensive part of GUC will be the creation of the mappings between schemas. This is however application-dependent, meaning that this is done only once per application, independent of the users. The user part of GUC only stores instance data for applications the user uses, and applies the available mappings to translate data from one schema to another. As users typically only work with a limited number of applications at the same time, this will not become a bottleneck on the system.

Another serious concern is privacy and authorization. It is important to get from users an *informed consent* about what data is communicated to what applications. This should lead to a controlled environment that addresses issues like identification, authorization and trust. The GUC data manager provides a point were all user data can be controlled and privacy policies can be applied, both on application and data element level. Privacy and trust is a research field in its own right. This part will be researched in the AlterEgo-project [16] we are involved in.

6 Conclusion and Future Work

In this paper we introduced a new generic component for the storage of user models and the exchange of these user models between applications. This generic component, called GUC, is part of a series of generic components for application development. We described the GUC high-level architecture in a number of steps. First, we described the basic functionality as a user model repository. Then, we discussed the need for schema mapping in order to exchange data between application user models. Finally, we described the integrated architecture, which provides the functionality in GUC to share and exchange user models between applications. We also explained that GUC is very easy configurable for different applications and discussed a number of configurations to use GUC in several scenarios.

The obvious next step is to continue the implementation of the presented GUC architecture. Within the implementation we emphasize on flexibility in the configuration so that we can easily choose for a specific functionality for the different functions that we identified. Furthermore we want to connect this component to the other components that we implemented to thus use it in the construction of a complete functional AWIS.

References

[1] Orwant, J.: Heterogeneous learning in the doppelgänger user modeling system. In: User Modeling and User-Adapted Interaction 4(2). (1995) 107–130

[2] Heckmann, D., Krger, A.: A user modeling markup language (userml) for ubiquitous computing. In: User Modeling 2003, Springer (2003) 393–397

[3] Heckmann, D., Schwartz, T., Brandherm, B., von Wilamowitz-Moellendorff, M.: Gumo, the general user model ontology. In: To appear in: User Modeling 2005, Springer (2005)

[4] Brooks, C., Winter, M., Greer, J., Gord, M.: The massive user modelling system (mums). In: Proceedings of the Seventh International Conference on Intelligent Tutoring Systems, Springer (2004) 635–645

[5] Kobsa, A.: Generic user modeling systems. User Modeling and User-Adapted Interaction **11** (2001) 49–63

[6] Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering semantic web information systems in Hera. Journal of Web Engineering, Rinton Press **2** (2003) 003–026

[7] Fiala, Z., Hinz, M., Meiner, K., Wehner, F.: A component-based approach for adaptive dynamic web documents. Journal of Web Engineering, Rinton Press **2** (2003) 058–073

[8] Fiala, Z., Houben, G.J.: A generic trancoding tool for making web applications adaptive, To appear in: The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal (2005)

[9] Houben, G.J., Fiala, Z., van der Sluijs, K., Hinz, M.: Hera meets AMACONT - SAG prototype. http://www-mmt.inf.tu-dresden.de:8081/sag/index.html. (2005)

[10] McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/. (February 2004)

[11] Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), IEEE Computer Society (2002) 117–129

[12] Doan, A., Domingos, P., Halevy, A.: Ontology matching: A machine learning approach. In: Handbook on Ontologies, S. Staab, R. Studer (Eds.). International Handbooks on Information Systems, Springer (2004) 384–405

[13] Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10** (2001) 334–350

[14] Broekstra, J., Kampman, A., van Harmelen, F.: A generic architecture for storing and querying rdf and rdf schema. The Semantic Web - ISWC 2002, number 2342 in Lecture Notes in Computer Science (2002) 54–68

[15] https://www.ist-mobilife.org/: MobiLife. (2005)

[16] Schuurmans, J., Zijlstra, E.: Towards a continuous personalization experience. In: Proceedings of the conference on Dutch directions in HCI, New York, NY, USA, ACM Press (2004) 19