Managing Personal Information through Information Components

Stefania Leone, Matthias Geel, and Moira C. Norrie

Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{leone|geel|norrie}@inf.ethz.ch

Abstract. We introduce the concept of information components and show how it can allow non-expert users to construct personal information spaces by selecting, customising and composing components defined by the system or other users. The system presented is based on a plug-and-play concept and new user-defined applications can be integrated into the portal-style interface based on default templates which can easily be customised by the users.
Keywords: information components, personal information management, pluggable interface architectures

1 Introduction

The term Web 2.0 refers to a new generation of Web-based applications that empower the user in the creation and management of content and services. Combined with the concepts of portals, widgets and mashups, users are nowadays able to not only manage and share their own data, but also integrate a wide range of external data and services. At the same time, users are encouraged to collaborate in a range of ways—including the community-based development of application libraries offered by sites such as Facebook.

It is therefore not surprising that users are increasingly turning to Web 2.0 sites for the management of personal information. However, this can in turn create its own problems in terms of losing control of one's own data and increased fragmentation of information across a range of Web 2.0 applications and desktop applications. At the same time, while sites such as Facebook provide a very large collection of applications for the management of personal information such as contacts, photo albums, places visited and virtual bookshelves, it is not possible to personalise these or combine them in flexible ways.

Our goal was to adopt concepts from Web 2.0 to empower users in the management of all of their personal information by allowing them to customise and compose *information components*. Instead of offering units of reuse at the interface or service level, we offer them at the database level, thereby allowing users to focus on their data requirements and to extend, customise, group or associate data items as they choose. A Web-based pluggable interface architecture generates the interfaces automatically based on default or selected templates which can easily be customised.

Section 2 provides the background to our work and an overview of the approach is given in Sect. 3. Section 4 describes the development process, while implementation details are given in Sect. 5. Our approach is compared to related work in Sect. 6. Concluding remarks are given in Sect. 7.

2 Background

Personal information management (PIM) systems proposed in research tend to either use a predefined PIM domain model e.g. [1, 2], or work according to a "no-schema" or "schema-later" approach e.g. [3, 4]. On the interface level, they mostly offer a generic browser where a user can browse via associations. In Haystack [4], entities are self-rendering in that they know how to render and display themselves and offer a context menu with the operations for that entity. While these systems help users work with information fragmented across existing applications, they do not provide the basic infrastructure to enable users to easily design, build and customise their own personal information space.

Web 2.0 has had a tremendous impact in terms of how people use the Web to communicate, collaborate and manage personal data. Its success can provide valuable lessons in how to provide easy-to-use, customisable and extensible platforms for PIM. In particular, users have become familiar with the plug-and-play style of interfaces offered by portals as well as applications offered by social networking sites such as Facebook. They are also becoming increasingly familiar with the notions of widgets that allow small, lightweight applications to be integrated into Web pages and Web-based mashups that allow the integration of services within a client. Taken together with the notions of user-generated content underlying Web 2.0, users are increasingly becoming empowered to manage their own information needs. However, on the negative side, the increased usage of Web 2.0 applications for PIM has drastic consequences in terms of loss of user control over their own data and also information fragmentation [5, 6].

We have adopted features of Web 2.0 for a PIM platform that allows users to define and manage their own personal information space by creating, sharing, customising and composing so-called *information components*. These components define the data structures, application logic and interaction logic that support particular information management tasks. Being able to build a personal information space in a plug-and-play manner through the selection and composition of components allows even non-expert users to profit from the experience of more advanced users and have fine-grained control over their PIM.

3 Approach

Information components are intended to be the basic units of reuse in information systems at both the schema and data level. An information space consists of a set of information components where each component can contain both metadata and data. If an information component consists of only metadata, reuse is only at the schema level to support the design of an information space. Optionally, an information component may contain data as well as metadata which allows the reuse of data. Information components can be composed from other components as shown in Fig 1.

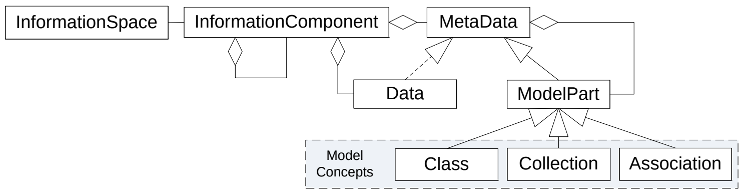

Fig. 1. Information components metamodel

Our prototype was developed using an object database based on the OM model [7] and therefore the metadata of a component is defined in terms of the model primitives which are classes, collections and associations. We show this in Fig 1 as a particular set of model parts to indicate that our notion of information components could be applied to other models.

Our approach allows users to construct their personal information space by defining their own components through a process of selecting, extending and combining existing components. A core set of system-defined components are provided to support the basic, common information management tasks in PIM systems such as the management of contacts and we show in the next section how a user can use these as the starting point for developing their own PIM applications. In addition, users can also reuse components defined by other users based on a global component registry.

An application consists of an information component together with an associated user interface (UI). To create an application, the user basically models the application domain and associates domain concepts to templates. The system then generates both the database representation of the domain model and the user interface based on the domain concept-template assignment specified by the user and deploys the application into the user interface.

4 Application Development Process

Assume a user has a simple contact application and a picture management application and would like to tag pictures with contacts. We will now illustrate how the user could extend their personal information space with this functionality by creating a new information component from the composition of the existing components. Figure 2 gives an overview of the steps involved as supported by the Application Manager integrated into our prototype system PIM 2.0.

① shows the PIM 2.0 UI with three applications. In order to create a new application, a user first creates a new information component ② by modeling the application domain reusing existing component parts and/or specifying new domain concepts. Note that an information component can have arbitrarily complex models, but may also represent a single domain concept. Figure 3 shows a screenshot of component composition in the Application Manager.

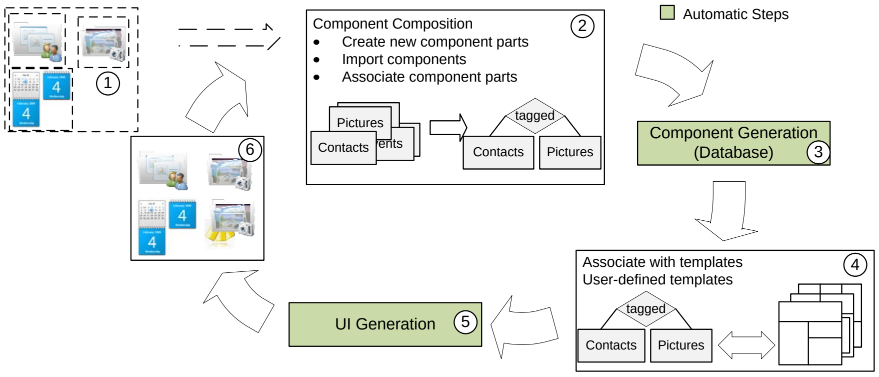

Fig. 2. Process of composing information components

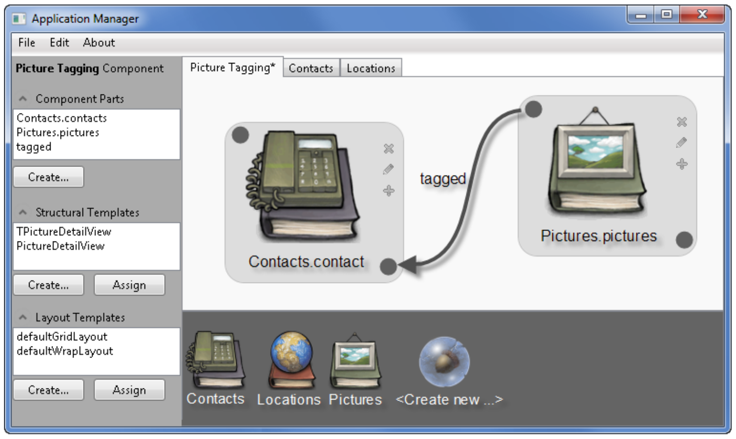

Fig. 3. Application Manager

We assume the reuse of a *Contacts* component and a *Pictures* component. The following assumes a combined definition of collections and types.

```
Contacts{
    contacts(name:string, birthday:date, phone:set, address:set)
}
Pictures{
    pictures(picture:uri,  caption:string)
    albums(name:string)
    picturesInAlbums(picture:ref, album:ref)
}
```

Users can drag and drop component parts into the composer area of the Application Manager to reuse them. New classes, collections and associations can be created by using the menu on the left. As shown in Fig. 3, a *PictureTagging* component can be created by reusing `contacts` from the *Contacts* component and `pictures` from the *Pictures* component and associating them with a `tagged` association. The textual representation of the new component is:

```
PictureTagging{
    Contacts.contacts(name:string, birthday:date, phone:set, address:set)
    Pictures.pictures(picture:uri, caption:string)
    tagged(picture:ref, contact:ref)
}
```

Collection and association names are qualified with their component names to ensure uniqueness. However, users can rename objects during composition in order to simplify the interface of the new component. Once defined, the component model is automatically created in the database ③.

To create the UI for the new component, it is associated with a structural template ④ which defines what should be displayed in terms of attributes and associated entities, the order in which these should be displayed and also the operations offered through context menus and buttons. To support the definition of the UI, there are standard templates for displaying collections and objects in read or write modes. Further, structural templates can be created automatically by the Application Manager by users defining views through the simple selection and ordering of object attributes. Also, context menus to select from various standard options are available where appropriate.

The actual layout and positioning of data is defined in separate layout templates that are applied upon interface generation. Note that users can create their own layout templates or extend existing ones. During UI generation, a view is generated which includes the layout as well as the structural information and represents the actual UI through which the user interacts with the data.

The default collection template displays a collection as a list and the user has to specify the object attributes to appear in that list. For example, they might specify that the `contacts` collection be displayed as a list of surnames followed by forenames. By default, collection views are always in write mode, so that objects can be selected, added to and removed from the list.

The default structural templates represent objects in a generic way. It is easy for users to create their own custom templates. For example, they might specify that in the picture detail view, the actual picture together with a caption should be displayed rather than the URL. To support such customisations, templates can be created which specify a set of applicable types and users are presented with a choice of presentations. The picture detail template would have the form:

```
<view name='PictureDetailView' mode='read'>
  <layout source='defaultGridLayout'>
  <compatibletypes>
    <type name='pictures'/>
  </compatibletypes>
  <attributes>
    <attribute name='picture' resource='attributes/picture'/>
    <attribute name='caption' value='attributes/caption/'>
  </attributes>
</view>
```

Attribute `picture` is a resource which means that the value is the path to the resource to be displayed, while the attribute caption is a value that can displayed directly. This template uses a default layout template *defaultGridLayout*, but a user can also define their own layout template or extend the default ones.

A user may wish to reuse the customised templates of imported components, extending them to cater for new data or functionality. For example, in the picture tagging application, the user might want to display the names of tagged persons along with the picture and caption. This could be done by creating a template that extends the `PictureDetailView` template as follows:

```
<view name=‘‘TPictureDetailView’’ extends name=‘‘PictureDetailView’’>
  ...
  <attributes>
    <attribute name = ‘Tagged’
       value=‘associations/tagged/contacts/attributes/name’ type=‘set’>
  </attributes>
</view>
```

The previous template is extended with an attribute that provides a set of names of the people related by the ‘tagged’ association of the *PictureTagging* component as specified by a path expression. By declaring `type=‘set’`, we indicate that the navigation may yield a set of values all of which should be displayed. After associating the model with the templates, the actual views are generated by combining the structural and layout templates ⑤. The application is then deployed into the portal. In our example, the PIM portal is extended and features the additional picture tagging application ⑥.

5 Implementation

The first step of implementing the PIM 2.0 system was to implement an object database that supported the concept of information components. In a second step, we implemented a Web application on top of the database that allows users to manage their personal information space by creating, composing and accessing information components through a portal-style interface.

Figure 4 provides an overview of the PIM 2.0 architecture. The system has two main parts—the Interface Manager and the Component Manager. The Component Manager is responsible for the creation and manipulation of information components as well as the management of the template assignments. The Interface Manager manages the template repository where the default structural and layout templates as well as user-defined templates are stored. Structural templates are written in an implementation-independent XML dialect whereas layout templates are written in an implementation-dependent manner since they are part of the underlying UI technology.

The component manager is implemented as a metamodel extension module [8] of our object database Avon [9]. A metamodel extension module consists of three parts, namely metamodel concepts, CRUD classes to manipulate the metamodel concepts and an optional language extension. Through the metamodel extension mechanism, the information component concepts are represented as metadata in the database.

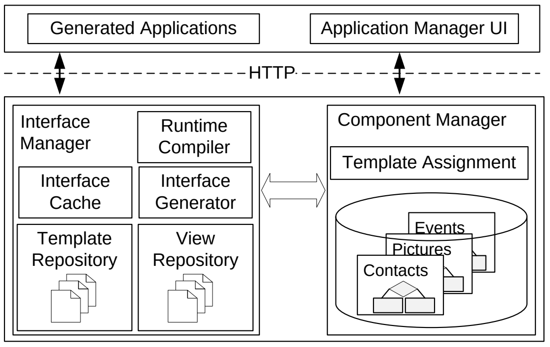

Fig. 4. Structure of PIM 2.0 system

The Web client offers a portal interface to a user's PIM system, where components are represented as portal applications. The Application Manager is also a portal application embedded within the PIM interface. The UI was implemented using the OpenLaszlo Web application framework. The OpenLaszlo architecture allows Web applications to be specified using a declarative XML syntax, Openlaszlo XML (LZX), that is then compiled to Flash on-the-fly. As an immediate benefit, this architecture allows us to compile automatically generated OpenLaszlo applications dynamically at runtime. We make use of this functionality to automatically load newly created applications into the PIM interface upon invocation of the view generation process on the server side.

6 Discussion

Our approach combines the advantages of predefined, no-schema and schema-later approaches to PIM by offering users a set of PIM components that can either be imported from a global registry or are already present in a user's local information space. The user can then extend or compose these to create new information components according to their information needs as they evolve.

While reuse in databases has been considered at the architectural level in terms of *c*omponent database systems [10] and also at the data level in terms of various forms of data integration services [11], little attention has been given to reuse at the database schema level to support reuse in the design and development of applications. An exception is the work of Thalheim [12] where he proposed the use of composable sub schematas to enhance the management of large and complex database schemas. In contrast, we focus on reuse as a means of allowing non-expert users to create a customised personal information space. We achieve this by providing them with a Web-based pluggable interface with an embedded set of graphical tools that enables them to create their personal information space through the selection, customisation and composition of components that are usually small and simple.

Our studies of existing PIM systems and also various Web 2.0 platforms such as Facebook shows that PIM application schemas tend to be rather small and simple. Users therefore tend to find PIM schemas easy to understand and our initial experiences with PIM 2.0 suggest that they have no problems to work with and compose our information components. However, this is something that requires detailed studies in the future.

7 Conclusion

We have presented the concept of information components as a mechanism for allowing users to construct their personal information space in a plug-and-play style of composing schemas and data. By supporting reuse within and across PIM systems, we believe that the more advanced Web users can create and share components with other users, while non-expert users can benefit from the expertise and experience of the community similar to collaboration evident in many Web 2.0 communities.

References

1. Gemmell, J., Bell, G., Lueder, R.: MyLifeBits: a Personal Database for Everything. Comm. ACM **49**(1) (2006)
2. Dong, X., Halevy, A.Y.: A Platform for Personal Information Management and Integration. In: Proc. CIDR 2005, Asilomar, CA, USA (2005)
3. Vaz Salles, M.A., Dittrich, J.P., Karakashian, S.K., Girard, O.R., Blunschi, L.: iTrails: Pay-As-You-Go Information Integration in Dataspaces. In: Proc. VLDB 2007, Vienna, Austria (2007)
4. Karger, D.R., Bakshi, K., Huynh, D., Quan, D., Sinha, V.: Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In: Proc. CIDR 2005, Asilomar, CA, USA (2005)
5. Leone, S., Grossniklaus, M., Norrie, M.C.: Architecture for Integrating Desktop and Web 2.0 Data Management. In: Proc. IWOOST 2008
6. Norrie, M.C.: PIM Meets Web 2.0. In: Proc. ER 2008, Barcelona, Spain (2008)
7. Norrie, M.C.: An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In: Proc. ER 1993, Arlington, Texas, USA (1993)
8. Grossniklaus, M., Leone, S., de Spindler, A., Norrie, M.C.: Dynamic Metamodel Extension Modules to Support Adaptive Data Management. In: Proc. CAISE 2010, Hammamet, Tunesia (2010)
9. Norrie, M.C., Grossniklaus, M., Decurtins, C., de Spindler, A., Vancea, A., Leone, S.: Semantic Data Management for db4o. In: Proc. ICOODB, Berlin, Germany (2009)
10. Dittrich, K.R., Geppert, A., eds.: Component Database Systems. Morgan Kaufmann (2001)
11. Halevy, A., Rajaraman, A., Ordille, J.: Data Integration: the Teenage Years. In: Proc. VLDB 2006, Seoul, Korea (2006)
12. Thalheim, B.: Component Development and Construction for Database Design. Data & Knowledge Engineering **54**(1) (2005)