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 their 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 where new user-defined applications can be created and 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, some users have evolved to so-called advanced Web users [1] that not only manage and share their own data, but also integrate a wide range of external data and services. At the same time, they 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. To demonstrate the feasibility of the approach, we

P. Soffer and E. Proper (Eds.): CAiSE Forum 2010, LNBIP 72, pp. 1–14, 2010.

have developed the PIM 2.0 platform based on an extension of an object database system that supports the information component concept. PIM 2.0 offers a Web-based pluggable interface architecture capable of automatically generating the portal-style interfaces based on default or selected templates which can easily be customised.

In this paper, we will focus on the technological level of our approach. We begin in Sect. 2 with a discussion of the background before introducing our notion of information components in Sect. 3. We then present our approach by first describing the process of extending a personal information space through the composition of components in Sect. 4 and then discussing the system architecture in Sect. 5. Implementation details of our PIM 2.0 system are given in Sect. 6. A discussion of the current status of our work and future directions of research is given in Sect. 7. Concluding remarks are given in Sect. 8.

2 Background

With the dramatic increase in the volume of personal data stored by users in recent years, there have been various proposals to build tools on top of existing file systems and desktop applications to extract and integrate data from different sources in order to meet a user's information needs, e.g. [2,3]. In the position paper by Franklin, Halevy and Maier [4], they propose a notion of *dataspace systems* where traditional database functionality such as metadata management, indexing and query processing can be used alongside traditional file systems and applications to support the administration, discovery and enhancement of personal data. Personal information management (PIM) solutions based on this vision aim to integrate all of a user's data and provide support for associating data of various types and from various sources, mostly following the vision of trails proposed in [5]. These associations may be automatically created as well as user-defined [3,6].

Generally, PIM systems proposed in research tend to either use a predefined PIM domain model e.g. [7,3,8], or work according to a "no-schema" or "schema-later" approach e.g. [6,9,2]. On the interface level, they mostly offer a generic browser where a user can browse data via associations. In Haystack [2], 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. We believe that, with the right tools, users could be put in control of what data they manage and how they manage it. Our general aim is not dissimilar to that of MashArt [1,10] where they provide a platform that enables advanced Web users rather than developers to create their own applications through the composition of user interface (UI), application and data components. The main difference is that they focus on the integration of existing services, whereas we want to empower users in the creation as well as the composition of components. Further,

our components are *information components* rather than *services* and integration is done at the database level. Therefore, whereas MashArt uses event-based composition, the composition of information components involves creating new data structures that integrate and augment existing metadata and data. Last but not least, we support the automatic generation of the UI based on a portal-style of interface familiar from Web 2.0 applications.

Web 2.0 has already 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 the plug-and-play style of applications offered by social networking sites such as Facebook which now offers thousands of applications developed by the user community. 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's control over their own data and also information fragmentation [11,12].

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

While reuse in databases has been considered at the architectural level in terms of component database systems [13] and also at the data level in terms of various forms of data integration services [14], 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 [15] 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. We note that in the case of object databases where the database schema corresponds to class definitions that include application logic and also possibly presentation and interaction logic, schema reuse can have a significant impact. However, we believe that reuse is

important even for relational or XML databases with no application logic integrated into the schema to enable non-expert users to design and develop their own personal information spaces.

3 Approach

Information components are intended to be the basic units of reuse in information systems at both the schema and data level. Fig. 1 gives an overview of the information component meta model. An information space consists of a set of information components where each information component can define both metadata and data. An information component may reuse data and metadata from other components and may expose data and metadata for reuse. The reuse is reflected by the import aggregation, which defines the structures that have been imported from other components. In turn, the export aggregation specifies the data and metadata that are shared for reuse in new components. When creating a new component, one can import and reuse structures from existing components, if desired. If an information component exports only metadata, reuse is only at the schema level to support the design of an information space. Optionally, an information component may export data in addition to metadata which allows the reuse of data.

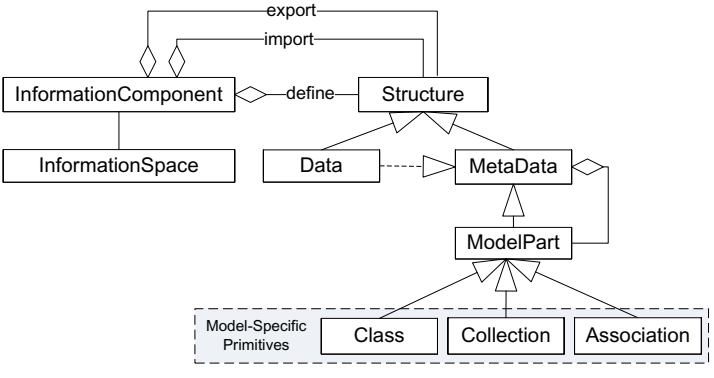

Fig. 1. Information components metamodel

Our prototype was developed using an object database based on the OM model [16] 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 applications defined by other users based on a global registry.

4 Application Development Process

Having introduced the concept of information components, we now describe how this can be used to build a personal information space through the composition of information components. 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.

An application consists of an information component together with an associated user interface. Figure 2 gives an overview of the steps involved in building an application. Before detailing the steps, we note that some tasks are carried out by the user while others are done automatically by the system. To create a new 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. Our PIM 2.0 system has a portal-like interface and we provide an Application Manager application embedded into the PIM portal that supports all of these operations.

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 modelling the application domain. A user may choose to reuse existing component parts and/or specify new domain concepts. Note that an information component can have arbitrarily complex models, but may also represent a single domain concept.

In our example, we assume the reuse of two information components, the *Contacts* component and the *Pictures* component. According to the constructs of the OM data model, a component will consist of one or more collections of a particular membertype plus associations between collections. Cardinality constraints can be specified over associations. In addition, the OM model can represent rich classification structures through the use of subcollection and subtyping relationships together with classification constraints such as disjoint and cover. It is beyond the scope of this paper to describe the OM model in full. Details of the model are given in [16].

In the following examples, we use an abbreviated form of the data definition language associated with the OM model for the sake of simplicity. In particular, we will specify the collections and their membertypes together, although we note that it is possible in the OM model to define multiple collections with the same membertype and therefore they are usually defined separately.

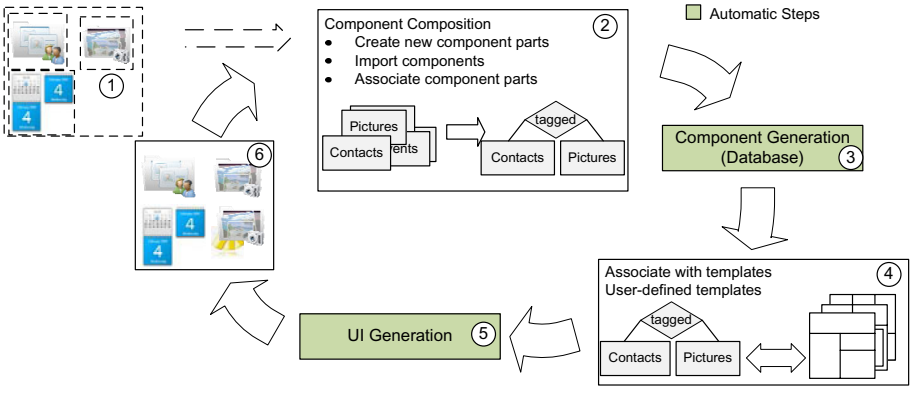

Fig. 2. Process of composing information components

Assume that the *Contacts* component has a single collection `contacts` defined as:

```
contacts(name:string, birthday:date, phone:set, address:set)
```

Further, assume that the *Pictures* component is defined as:

```
pictures(picture:uri, caption:string)
albums(name:string)
picturesInAlbums(picture:ref, album:ref)
```

where `picturesInAlbums` is an association between `pictures` and `albums`. To create a *PictureTagging* component, the user could choose to import `contacts` from the *Contacts* component and `pictures` from the *Pictures* component and associate these two parts with a `tagged` association ②, defined as follows:

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

Note that the names of collections, types, attributes etc. may be qualified with their component names to ensure uniqueness. However, we also provide options for the user to rename objects during the composition process in order to simplify the interface of the new component.

Figure 3 shows a screenshot of the component composition process in the Application Manager. 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 using the menu on the left. For the *PictureTagging* component, the `contacts` collection from the *Contacts* component and the `pictures` collection from the *Pictures* component have been associated with a `tagged` association in the composer area. Once defined, the component model is automatically created in the database ③.

To create the interface of the new component, the user can associate the component parts to structural templates that define how the data will be displayed ④. These templates define what should be displayed in terms of attributes

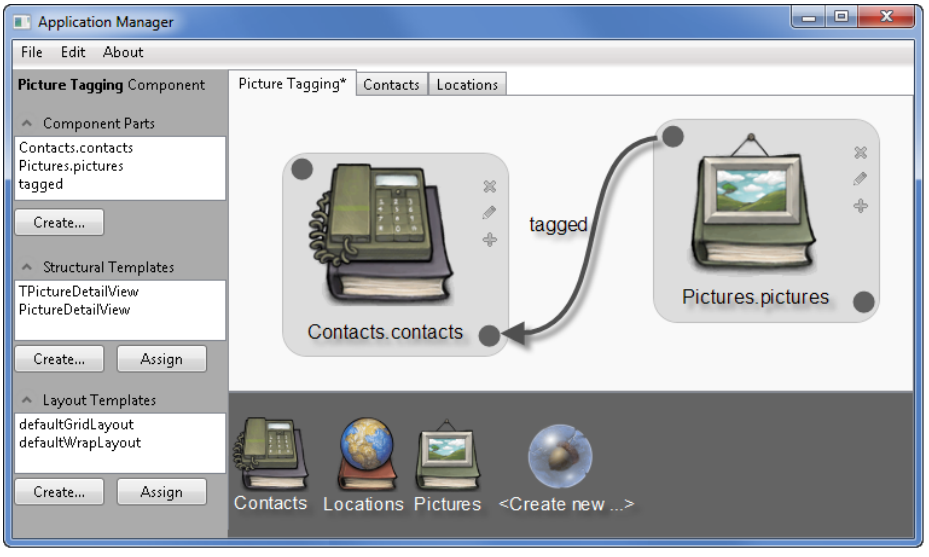

Fig. 3. Application Manager

and associated entities, the order in which these should be displayed and also the operations offered through context menus and buttons. To facilitate the process of defining the user interface, we defined standard templates for displaying collections and objects in read or write modes. Further, standard structural templates can be extended or new ones created through 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 be 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 Architecture

Having described the general process of composing information components, we now look in more detail at how our approach works in practice in terms of supporting the local and global reuse of components. We start by looking at the structure of the PIM 2.0 system which manages and provides access to a user's personal information space based on the concept of information components presented in the previous sections.

Figure 4 provides an overview of the PIM 2.0 structure. 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 specific technologies used and implementation details are given in the next section.

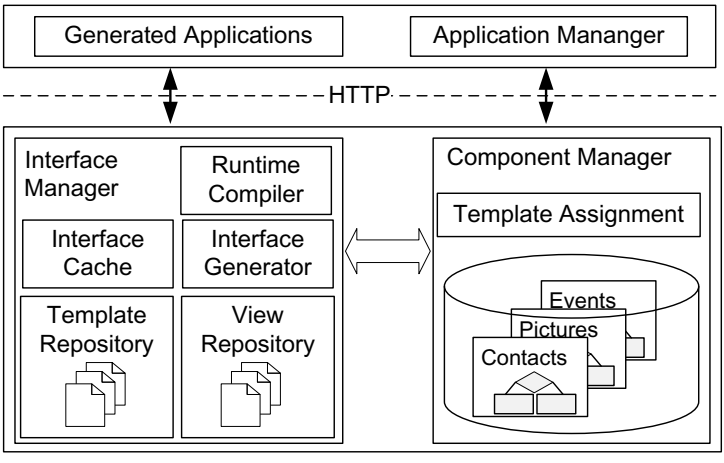

Fig. 4. Structure of PIM 2.0 system

The user interface consists of two parts, the actual portal-like user interface which is the set of applications to manage personal data and the Application Manager that allows users to create new applications by defining new information components. The Application Manager exhibits the functionality presented in Sect. 4, namely component design, template design and component-template assignment. As mentioned previously, the Application Manager is provided as an application embedded in a user's PIM portal.

Since there are many common information management tasks in PIM systems such as the management of contacts and todo lists, we offer a core set of system-defined components for these tasks. This means that the user can start by selecting the components that they require from this set. However, it is simple for them to create new customised components based on these by selecting which parts of the core components they want to use and/or extending them with additional information. We refer to this as *local reuse* and it often occurs as part of the evolution of a personal information space and not just the initial design. This means that it is common that not only the metadata, but also the data is integrated as part of the composition process. Data integration usually takes the form of creating new objects, collections or associations that enable data from the imported components to be aggregated or linked together.

While users could use the system to design and build their personal information space based on a set of pre-defined components, one of the central ideas behind our approach is that they can benefit from the experience and expertise of others through the reuse of components developed within the user community. We refer to this as *global reuse*. In addition to the local repository of components and associated templates shown in Fig. 4, there is a global registry where users can make their information components available to others. A component can be registered together with one or more associated structural and view templates. A user can therefore search for components and templates in the global registry and select among the variants offered for a particular information management task. Since information components are typically rather simple, it is not difficult for a user to select a component based on keywords, description and inspection. Clearly there are also issues related to trust and security and we are currently investigating a combination of technical approaches based on making PIM 2.0 a secure, controlled environment and ones of trust commonly adopted in Web 2.0 platforms such as Facebook.

6 Implementation

In this section, we describe how we implemented the PIM 2.0 system. The first step was to implement an object database system that supported the concept of information components defined in this paper. 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 5 gives an overview of our layered system architecture. For each layer, we indicate the technologies used. The database layer consists of an object database that has been extended with the information component concept. We have developed our own object database system, OMS Avon, which is based on the OM model and implemented in Java [17]. The information component concept is implemented as a meta model extension module [18]. Since OMS Avon bootstraps its meta model, we were able to extend the Avon meta model with the information component meta model and thus have the information component concepts as metadata concepts natively in the database.

In OMS Avon, meta model concepts are manipulated via corresponding CRUD classes. These implement the basic data management operations of create, read, update and delete for the various meta model concepts as well as providing additional methods that implement more sophisticated operations. We extended the set of CRUD classes for types, collections, associations and constraints provided by Avon with an information component CRUD class, that supports the creation, retrieval, deletion, composition and manipulation of information components. Upon creation, the `import` method supports the importation and thus composition of components, while the `export` method exposes component structures. The actual data is accessed via the `ObjectsCrud` class which supports the functionality of creating, retrieving and deleting objects, as well as retrieving and updating attribute values.

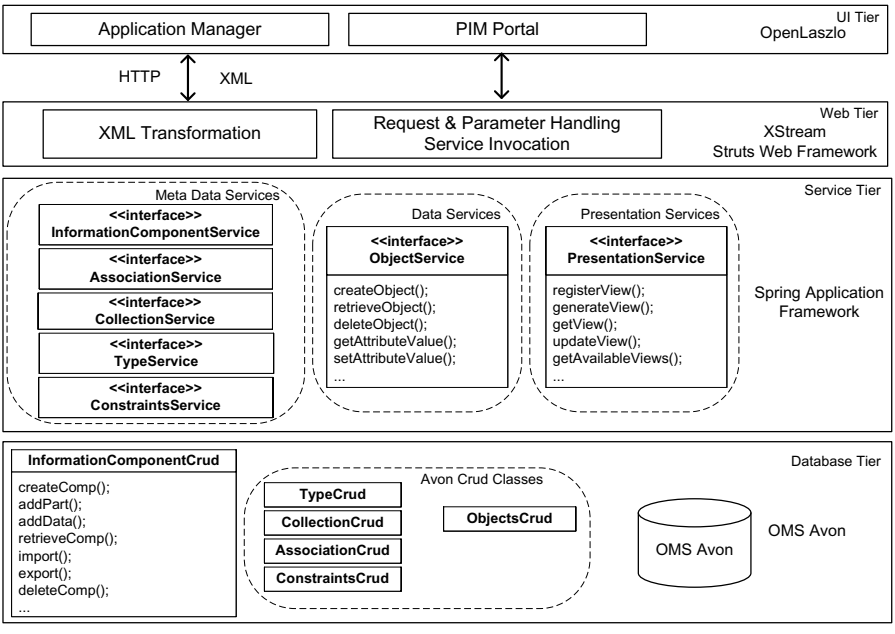

Fig. 5. Layered architecture of framework

These CRUD classes are used by the service tier to manage information components. The service tier exposes the information component concepts as a Web service. For each meta model concept, a service class has been created which offers methods to deal with these concepts. These service classes contain additional application logic needed by the Application Manager. Additionally, this tier introduces model classes that correspond to entities from the database schema such as collections and associations. These are used to transfer data between the server and client via the Web service interface. All model classes are simple Java classes and follow the JavaBeans specification. The service tier separates the services in three classes. *Metadata Services* provide methods to deal with information components and allow the definition of domain models using OM concepts. *Data Services* are used to create and manipulate domain objects i.e. the actual data. The *Presentation Service* offers methods to manage the view definitions and generation as well as functionality to register structural and layout templates for specific types.

The Web service tier mediates data between the service tier on the server-side and the Web-based UI on the client side. It has been decoupled from the service tier in order to allow for different Web service implementations. This tier processes URL requests to the Web service and serves responses back to the clients. It was implemented using the Struts Web application framework and uses XStream to marshal the Java model classes to XML and back.

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 and is used to create, compose and manage applications. The UI has been implemented using the OpenLaszlo Web application framework[1]. 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.

The Presentation Service on the server-side orchestrates the whole interface generation process and upon generation invocation lets the Interface Manager retrieve the required template files from the file system. It then invokes the View Generator that composes these templates and injects additional declarative statements where necessary, such as bindings of XML data to structural OpenLaszlo elements via XPath. The resulting OpenLaszlo files are then stored back to the file system. These OpenLaszlo XML files are ready to be compiled by OpenLaszlo as soon as they are requested for the first time.

7 Discussion

Our ultimate goal is to enable users to design and build their own personal information spaces according to their specific information requirements. To achieve this, we propose an approach that allows users to construct their personal information space in an incremental way based on the reuse and composition of both data and metadata shared with other users. These ideas are similar to the idea of crowdsourced systems [19], where members of the user community develop and share applications that can be reused by other members of the community. While ideally regular users would be supported in composing their information spaces, in reality, our system addresses the needs of advanced Web users who are familiar with Web 2.0 platforms and have worked with technologies such as widgets and mashups.

While most PIM systems are either based on predefined, no-schema or schema-later approaches, we combine the advantages of all of these and offer a user a set of existing 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. 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.

Metadata and data reuse is currently done by reference in the case of local reuse and by copy in the case of global reuse. In earlier work [11], we investigated the

[1] `http://www.openlaszlo.org/`

problem of data replication across a user's information space and Web 2.0 applications and proposed a framework that supports the synchronisation of local data with replications of that data on social networking sites. This is closely related to the importation of global components and the desire to stay 'synchronised', especially when importing not only schema, but also data. For example, one could import a movie database from a global provider such as IMDB, where a user extends the movie database with personal ratings and comments. A user, however, would want to have an up-to-date movie database, which asks for periodical synchronisation with the original source. We are currently investigating different options of how to support reuse within and across information spaces.

While our prototype implementation of PIM 2.0 has shown the feasibility of our approach within a user's information space, we are currently investigating several issues related to global reuse. As already mentioned, one area of research is to investigate issues of security and trust. While we believe that it is important to have mechanisms that ensure for example that users' personal data cannot be accessed by other parties unless explicitly shared, we also think it is important to consider the notions of trust common in Web 2.0 platforms and the extent to which these can be applied in more general settings of PIM.

Finally, we report on our experiences concerning the particular choice of technologies used to implement PIM 2.0. As described earlier, we used OpenLaszlo for the UI of our system. OpenLaszlo is one example of a Rich Internet Application (RIA) framework and we have also experimented with alternatives. Our experience has shown that while RIA frameworks are well suited to rapid prototyping, they are not as flexible and dynamic as expected, especially in terms of special purpose UI widgets and non-standard interaction models that are not inherent in the system. As a result, while OpenLaszlo enabled us to successfully demonstrate the concept, the current version of PIM 2.0 is rather limited in terms of rich user interaction and therefore not really suitable for carrying out detailed user studies. We are therefore investigating alternative implementation strategies.

8 Conclusion

We argue that users should be empowered in the management of their personal data by providing tools that can enable them to design and build PIM systems adapted to their personal information requirements. 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 advanced Web users can create and share components with other users, while non-expert users can still benefit from the expertise and experience of the community similar to collaboration evident in many Web 2.0 communities.

References

1. Daniel, F., Casati, F., Benatallah, B., Shan, M.C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
2. Karger, D.R.: Haystack: Per-User Information Environments. In: Kaptelinin, V., Czerwinski, M. (eds.) Beyond the Desktop Metaphor: Designing Integrated Digital Work Environments, 1st edn., vol. 1, pp. 49–100. The MIT Press, Cambridge (2007)
3. Dong, X., Halevy, A.Y.: A Platform for Personal Information Management and Integration. In: Proc. CIDR 2005, Asilomar, CA, USA (2005)
4. Franklin, M., Halevy, A., Maier, D.: From Databases to Dataspaces: A New Abstraction for Information Management. ACM SIGMOD Record (December 2005)
5. Bush, V.: As We May Think. The Atlantic Monthly 176(1) (1945)
6. 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)
7. Gemmell, J., Bell, G., Lueder, R.: MyLifeBits: a Personal Database for Everything. ACM Comm. 49(1) (2006)
8. Shoens, K.A., Luniewski, A., Schwarz, P.M., Stamos, J.W., Thomas, J.: The Rufus System: Information Organization for Semi-Structured Data. In: Proc. VLDB 1993, Dublin, Ireland (1993)
9. Freeman, E., Gelernter, D.: Lifestreams: a Storage Model for Personal Data. SIGMOD Record 25(1) (1996)
10. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing 11(3) (2007)
11. Leone, S., Grossniklaus, M., Norrie, M.C.: Architecture for Integrating Desktop and Web 2.0 Data Management. In: Proc. IWOOST 2008, Yorktown Heights, USA (2008)
12. Norrie, M.C.: PIM Meets Web 2.0. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 15–25. Springer, Heidelberg (2008)
13. Dittrich, K.R., Geppert, A. (eds.): Component Database Systems. Morgan Kaufmann, San Francisco (2001)
14. Halevy, A., Rajaraman, A., Ordille, J.: Data Integration: the Teenage Years. In: Proc. VLDB 2006, Seoul, Korea (2006)
15. Thalheim, B.: Component Development and Construction for Database Design. Data & Knowledge Engineering 54(1) (2005)
16. Norrie, M.C.: An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In: Elmasri, R.A., Kouramajian, V., Thalheim, B. (eds.) ER 1993. LNCS, vol. 823. Springer, Heidelberg (1994)
17. Norrie, M.C., Grossniklaus, M., Decurtins, C., de Spindler, A., Vancea, A., Leone, S.: Semantic Data Management for db4o. In: Proc. ICOODB 2009, Berlin, Germany (2009)
18. Grossniklaus, M., Leone, S., de Spindler, A., Norrie, M.C.: Dynamic Metamodel Extension Modules to Support Adaptive Data Management. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 363–377. Springer, Heidelberg (2010)
19. Kazman, R., Chen, H.M.: The Metropolis Model a New Logic for Development of Crowdsourced Systems. ACM Commun. 52(7), 76–84 (2009)