Ersch. in: Web information systems engineering : 11th international conference ; proceedings / WISE 2010, Hong Kong, China, December 12 - 14, 2010 / Lei Chen, Peter Triantafillou, Torsten Suel (eds.). - Berlin [u.a.] : Springer, 2010. - S. 411-418. - (Lecture notes in computer science ; 6488). - ISBN 978-3-642-17615-9

http://dx.doi.org/10.1007/978-3-642-17616-6_37

Synchronising Personal Data with Web 2.0 Data Sources

Stefania Leone, Michael Grossniklaus,
Alexandre de Spindler, and Moira C. Norrie

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

Abstract. Web 2.0 users may publish a rich variety of personal data to a number of sites by uploading personal desktop data or actually creating it on the Web 2.0 site. We present a framework and tools that address the resulting problems of information fragmentation and fragility by providing users with fine grain control over the processes of publishing and importing Web 2.0 data.

1 Introduction

An increasing amount of personal data is being published on Web 2.0 sites such as Facebook and Flickr as well as various forms of discussion forums and blogs. These sites provide easy-to-use solutions for sharing data with friends and user communities, and consumers of that data can contribute to the body of information through actions such as tagging, linking and writing comments. Sometimes, Web 2.0 sites also take over the primary role of managing a user's personal data by providing simple tools for creating, storing, organising and retrieving data, even when users are on the move.

Desktop applications for multimedia processing such as Adobe Photoshop Lightroom provide tools to support the publishing process. There is also an increasing number of tools to support cross publishing, meaning that information posted on one site is published automatically to another. For example, WordPress allows information published on Twitter to be automatically published as a blog article and vice versa. Although such tools facilitate the publishing process, they are only available for certain types of data and applications. Further, since it is now typical for active users of Web 2.0 sites to publish data to many different sites, they need to learn to use a variety of publishing tools, some of which require manual activation while others are fully automated.

Another problem is the *fragility* of information managed solely by Web 2.0 sites since many of these sites are considered to have an uncertain future. For example, the expenses incurred by YouTube far exceed their advertisement revenues which has led to speculation about its future [1]. It may therefore be desirable to *import* data from Web 2.0 sites into desktop applications to ensure offline and/or long-term access.

To address these issues, there is a need for a single, general tool to support the processes of publishing and importing Web 2.0 data across applications. It should be easy to specify new synchronisation processes and data mappings through a graphical user interface. In addition, it should be possible to integrate existing tools for publishing and cross publishing, while allowing users to control the actual synchronisation processes. As a first step in this direction, we have developed a synchronisation framework which allows users to synchronise personal data with one or more Web 2.0 sites based in a plug-and-play style of selecting and configuring synchronisation modules.

Section 2 discusses the background to this work, while Sect. 3 provides an overview of our approach. Section 4 examines support for the required data mappings. Details of the synchronisation engine are given in Sect. 5. Concluding remarks are given in Sect. 6.

2 Background

Nowadays, it is common for users to manage a wide variety of personal information using a mix of desktop applications and Web 2.0 services. Data is often replicated with a user keeping a copy on the desktop as well as publishing it on one or more Web 2.0 sites. For example, a version of a photo may be published on Facebook and Flickr, and a friend on Facebook may also be connected to on LinkedIn. In other cases, the data is created and managed using the Web 2.0 service, leaving the user entirely dependent on that service for access to the data.

While Web 2.0 services offer many advantages over traditional desktop applications in terms of data sharing and ubiquitous access, there are also drawbacks resulting from the fragmented management of personal data across desktop applications and Web 2.0 services. To address this problem, researchers have proposed different ways of providing file system services over Web 2.0 data sources. One approach is to propose that Web 2.0 applications should use independent file system services [2] rather than managing their own data. A less radical approach is to propose a software layer that supports an integrated file system view of a user's data based on a common interface to Web 2.0 services [3]. In both cases, naming and security are major issues. There are also efforts in the industry sector to promote open protocols and standards such as DataPortability.org. OpenLink Data Spaces[1] (ODS) implements the recommendation from DataPortability.org and offers several components for personal data management such as an address book and picture gallery as well as integration with applications such as Facebook.

Other researchers have addressed the problem of extracting and aggregating data from Web 2.0 services, for example [4–6]. While these approaches mostly use data extraction techniques for gathering data, many Web 2.0 platforms nowadays provide an API which provides access to platform data. We note that a number of commercial data aggregator services are available for news, blogs and social

[1] `http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/Ods`

networking data, e.g. FriendFeed[2]. Content aggregator services such as GNIP[3] act as an intermediate data aggregator between Web 2.0 services and applications by offering developers a uniform API and polling services.

While these projects provide partial solutions, they do not deal with issues of data replication and volatility. Also, they pay little attention to the publishing process or the need to import Web 2.0 data in order to ensure both offline and long-term access. Further, in cases where data synchronisation is supported, it tends to be hard-wired in the sense that the developer predefines data mappings and means of handling conflict resolution.

Web 2.0 platforms provide opportunities to address data synchronisation issues in a novel way by passing greater responsibilities, and hence more control, to the user. Just as users have become actively involved in, not only the creation and sharing of content, but also the development and integration of applications, they should be empowered to take decisions on what, when and how data should be synchronised in order to meet their own information needs. Therefore, rather than providing a fully automated and transparent solution, we believe that it is important to provide a flexible and configurable framework that gives users the freedom to either select synchronisation modules developed/configured by others or to develop/configure their own modules.

3 Synchronisation Framework

Our synchronisation framework allows users to control all synchronisation of personal data with Web 2.0 data in a single place. In Fig. 1, we present an overview of the architecture of the framework. It consists of a synchronisation engine, a number of synchronisation modules and some utilities, and exposes an API that offers the synchronisation functionality. A synchronisation module defines a particular synchronisation process in terms of internal and external synchronisation endpoints, a data mapping and a synchronisation algorithm.

The *synchronisation endpoints* are adapters to data sources and encapsulate the access to these sources. An external synchronisation endpoint is the conceptual representation of a Web 2.0 data source that includes information about the data source's data model as well as other synchronisation specific information such as access and authentication characteristics. We make use of the APIs offered by Web 2.0 applications such as Facebook and Flickr to implement synchronisation endpoints and access their data. These platforms usually expose their data and functionality through REST-based APIs. An internal synchronisation endpoint is the conceptual representation of a desktop application. The framework is extensible in that synchronisation with additional external and internal data sources can easily be implemented by adding an associated synchronisation endpoint.

The *synchronisation engine* is responsible for the actual synchronisation process. It first configures the synchronisation process based on the specification

[2] `http://friendfeed.com`
[3] `http://www.gnip.com/`

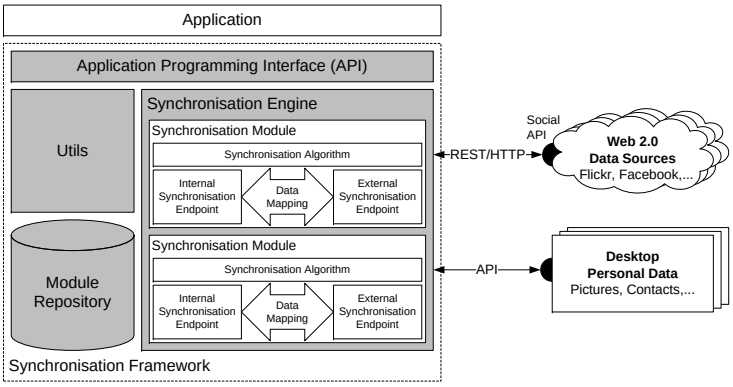

Fig. 1. Architecture of the synchronisation framework

provided by the synchronisation module and then executes it. The synchronisation module defines a mapping between the data models of the two data sources and also a synchronisation algorithm. Additionally, it specifies other configuration information such as when the synchronisation process should be executed and a conflict resolution strategy. Default mappings and specifications of the synchronisation processes are provided, but users can choose and configure a synchronisation module for a pair of endpoints or even develop their own.

Note that the distinction between internal and external endpoints is somewhat artificial, since the framework can be used to synchronise any two data sources. Thus, it could be used to support cross-publishing between two different Web 2.0 data sources such as Twitter and Facebook as well as between two desktop applications. However, since our main goal was to investigate support for publishing and importing Web 2.0 data, we have focussed so far on synchronising desktop data with social networking sites such as Facebook and Flickr.

4 Data Access and Data Mappings

The synchronisation endpoints provide access to Web 2.0 data sources through their APIs. While the Facebook API[4], Microsoft Live Services[5] and Flickr Services[6] are all platform-specific, the Google OpenSocial API[7] is a common API which can be implemented by any Web 2.0 service. Although several well-known social networking platforms, e.g. LinkedIn, implement the OpenSocial API, they usually implement only part of the API since many features are optional.

Some APIs only provide access to a platform's data while others allow applications or external data to be integrated. For example, Facebook allows users to publish their own applications developed using either the Facebook Markup

[4] `http://developers.facebook.com/`
[5] `http://dev.live.com`
[6] `http://www.flickr.com/services/api/`
[7] `http://code.google.com/apis/opensocial/`

Language (FBML) and Facebook Query Language (FQL) or a general programming language such as Java, PHP or Visual Basic. Similarly, services such as Orkut that implement the OpenSocial API can be extended by building gadgets using Google Gadget technology[8].

While Facebook and LinkedIn target different audiences and therefore offer different kinds of data and services, the data accessible through the APIs is actually quite similar in terms of the concepts and attributes. The main difference lies in "additional" attributes that can be associated with friends in the case of Facebook and business contacts for LinkedIn. Application developers are highly dependent on what the APIs offer in terms of functionality and data access. In the case of Facebook, a developer has read access to user profiles as well as to their immediate network of friends. However, the OpenSocial API offered by LinkedIn can only be used upon approval.

The data models for many Web 2.0 sites tend to be quite similar and also relatively simple and modular. Consequently, the data mappings between synchronisation endpoints tend to be much simpler than typical data integration scenarios tackled by the information systems community. This is something that we have been able to exploit in the data mapping tools provided for developers of synchronisation modules. We now describe how a data mapping between Outlook contacts and Facebook could be specified using our mapping tool.

Figure 2 shows parts of the two data models represented as ER models. The Outlook contact model shown on the left has a `Contact` entity which defines attributes such as `Full Name`, `Birth Date`, `Home Address` and `Business Address`. The central entity of the Facebook model shown on the right is `User` which defines attributes such as `Birthday`, `Name` and `HomeTown`. A user can be a member of one or more `Groups` and has a work history, which is a sequence of `WorkInfo` entities.

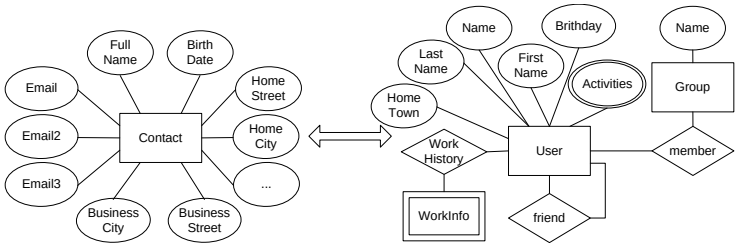

Fig. 2. Simplified Outlook and Facebook data models

The actual synchronisation of data is performed over data collections. To create a mapping between the data models, the collections of data to be synchronised must first be specified. Assume that we want to synchronise the collection of all contacts in Outlook with the collection of friends in Facebook. Using our mapping tool, we could define a mapping between the corresponding entity types

[8] `http://code.google.com/apis/gadgets/`

contact and user as shown in the screenshot in Fig. 3. The two data models are represented as tree structures, with the Outlook data model in the left window and that of Facebook on the right. Types and attributes can be mapped by simply dragging and dropping the tree nodes from one model over those of the other model. When creating such a mapping, the system checks whether the types of the two nodes are compatible. Basic transformations are done automatically by the system. For more complex transformations, the user has to select one of the provided data transformers.

The highlighted mapping defines a mapping between the attribute `fullName` of the local `contact` type to the attribute `name` of the Facebook `User` type. In the lower part of the screenshot, the navigation paths of that specific mapping are displayed. Note that the Facebook `User` type defines both attributes `firstName` and `lastName` as well as a composed `name` attribute. Since the local `contact` type only offers a composed `fullName` attribute, we chose the latter for the mapping. Attributes and entities which are mapped are tagged with a small lock label.

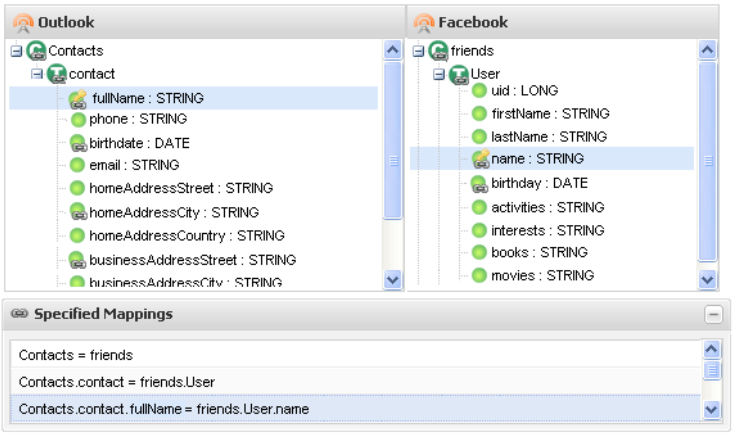

Fig. 3. Mapping tool screenshot

Structural mappings, such as mapping the first entity of Facebook's `Work-History` list to the business information in Outlook represented as a set of attributes defining an organisation's address, can also be performed.

5 Implementation

Figure 4 shows a simplified UML diagram of the system architecture. The synchronisation engine `SyncEngine` is the heart of the system and handles the synchronisation process. Parts of its functionality are extracted into configuration and synchronisation strategy objects allowing the engine to be adapted to the needs of an application and simplifying the implementation of extensions. A synchronisation strategy is a class that implements the interface `SyncStrategy`

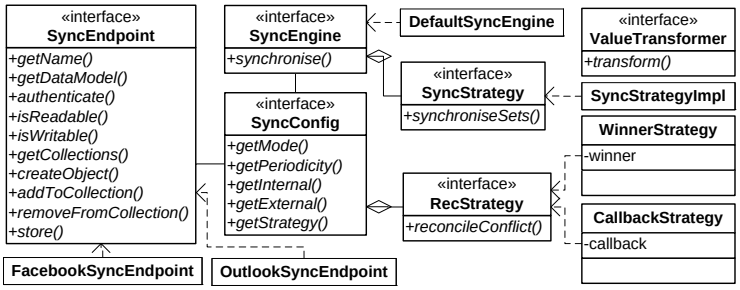

Fig. 4. Synchronisation framework UML diagram

and provides the actual synchronisation algorithm. We provide a naive synchronisation algorithm `SyncStrategyImpl` as default strategy but new strategies can easily be added by developing a class that implements the `SyncStrategy` interface.

`SyncEngine` uses `SyncConfig` to configure the synchronisation process. A `SyncConfig` object is associated with two `SyncEndpoint` objects `internal` and a reconciliation strategy object `RecStrategy` as well as configuration parameters such as `periodicity` and `mode`. Classes that implement the `RecStrategy` interface offer a strategy to handle conflicts that arise when synchronising. Currently, we provide two strategies. The `WinnerStrategy` can be configured to give priority to one of the endpoints, while the `CallbackStrategy` propagates the conflict resolution task to the user. Other conflict resolution strategies can be added by developing classes implementing the `RecStrategy` interface.

Synchronisation endpoints can be added by implementing the `SyncEndpoint` interface. An endpoint provides access to the data source's objects via one or more collections that group these objects. In our example, the Facebook endpoint offers access to a single collection `friends`, but a developer could use two collections, one for close friends and one for acquaintances.

Objects can be created as well as added to and removed from the actual collection to be synchronised. In order to synchronise data in an efficient way, synchronisation endpoints represent their data in a intermediate and uniform format based on object graphs following the JavaBeans conventions. For the data mappings specified through the user interface, we use OGNL (Object-Graph Navigation Language)[9]. Mapping expressions specify the mapping between the two models, both represented as object graphs with the collections to be synchronised as the root objects of the graphs. Mappings can either be created and manipulated using the visual mapping tool or through XML files. A simplified excerpt of a mapping for our example is shown below.

```
<cMapping iCollName="contacts" expr="friends"/>
<tMapping iName="contact" expr="User" key="fullName">
   <attrMapping iName="fullName" expr="name"/>
   <attrMapping iName="birthdate" expr="birthday"
      valueTransformerClass="..Birthdate2FBBirthday"/>
```

[9] `http://www.opensymphony.com/ognl/`

```
</tMapping>
```

`cMapping` maps the internal collection `contacts` to an external concept matching the OGNL expression `friends`. `tMapping` maps the internal type `contact` to a concept matching the expression `User` and defines its key to be `fullName`. There are two attribute mappings which map the local attributes `fullName` and `birthdate` to the expressions `name` and `birthday`, respectively. Both external expressions are evaluated by OGNL in the context of the the external `User` type. OGNL performs automatic type conversions between numeric types, booleans and strings. To support special conversions, we introduced value transformers that offer methods to convert values from one format to another. For example, the class `Birthdate2FBBirthday` transforms a birthdate value in the Outlook date format to the Facebook birthday format. Note that it is also possible to use transformer classes to convert from flat to nested structures and vice versa. Our framework provides several standard value transformer classes and additional transformers can easily be implemented.

6 Conclusion

We have presented a framework for synchronising personal data with Web 2.0 data sources that allows users to control where, when and how data is published to, and imported from, the Web. Tools are provided that enable users to configure the synchronisation process as well as to define new synchronisation processes along with the required data mappings. Some of the key distinguishing features of our approach stem from the fact that, instead of trying to develop new standards or architectures, our aim is to work with existing APIs and publishing services as well as taking into account the nature of data replication, data volatility and data inconsistencies evident in current Web 2.0 settings.

References

1. Wayne, B.: YouTube Is Doomed. The Business Insider (April 2009) `http://www.businessinsider.com/is-youtube-doomed-2009-4`.
2. Hsu, F., Chen, H.: Secure File System Services for Web 2.0 Applications. In: Proc. ACM Cloud Computing Security Workshop (CCSW 2009). (2009)
3. Geambasu, R., Cheung, C., Moshchuk, A., Gribble, S.D., Levy, H.M.: Organizing and Sharing Distributed Personal Web-Service Data. In: Proc. Intl. World Wide Web Conf. (WWW 2008). (2008)
4. Matsuo, Y., Hamasaki, M., Nakamura, Y., Nishimura, T., Hasida, K., Takeda, H., Mori, J., Bollegala, D., Ishizuka, M.: Spinning Multiple Social Networks for Semantic Web. In: Proc. Natl. Conf. on Artificial Intelligence (AAAI 2006). (2006)
5. Guy, I., Jacovi, M., Shahar, E., Meshulam, N., Soroka, V., Farrell, S.: Harvesting with SONAR: the Value of Aggregating Social Network Information. In: Proc. ACM Intl. Conf. on Human-Computer Interaction (CHI 2008). (2008) 1017–1026
6. Matsuo, Y., Mori, J., Hamasaki, M., Nishimura, T., Takeda, H., Hasida, K., Ishizuka, M.: POLYPHONET: an Advanced Social Network Extraction System from the Web. Web Semant. **5**(4) (2007) 262–278