Application Design and Interoperability for Managing Personal Information in the Semantic Desktop*

Huiyong Xiao Isabel F. Cruz

Department of Computer Science
University of Illinois at Chicago
{hxiao | ifc}@cs.uic.edu

Abstract. A number of *semantic desktop* frameworks have been proposed that address different issues in personal information management, including the organization, manipulation, and visualization of personal data. Our approach comprises a layered multi-ontology based framework, MOSE. In this paper, we describe how the architecture of MOSE supports an application centered semantic approach to personal information management. In particular, we introduce a method based on the Model-View-Controller paradigm for *personal information application* (PIA) development, as enabled by the underlying semantic data organization and specified by the *PIA designer*. We also introduce the notion of *parameterized channel* and describe how it supports the interaction among multiple visualizations in a PIA. We give a definition of *desktop service* based on the concept of parameterized channel and the benefits provided by the separation of PIA definition and implementation. Finally, we discuss two cases of the execution of desktop services in MOSE that highlight application interoperability.

1 Introduction

With the development of computer device manufacturing, storage is no longer a bottleneck for computer users. As users will not have to remove outdated data from storage, the increasingly large amount of data poses a critical problem: the organization of the personal information space so as to enable efficient and effective data management.

Typically, personal information in a desktop is characterized by: (1) Disparate models: the personal data tend to be of different formats and use different models for data storage and presentation. (2) Unstructured contents: in addition to structured data (e.g., a relational database in an Microsoft Access file) and to semi-structured data (e.g., XML files), most data is stored as unstructured (nontextual and textual) files. (3) Latent semantics: many types of files (e.g., a saved email message) present their contents in natural language, where the semantics of the data is latent and implicit (e.g., "my picture in the attachment" in a message referring to the attached picture of the sender). (4) Lack of semantic associations: existing desktop operating systems store data as files that are classified and organized in hierarchical directories.

The directory based model creates a limitation on the access to personal data and justifies the need for a semantically rich way of personal information organization. In

* This work was partially supported by NSF Awards ITR IIS-0326284 and IIS-0513553.

1945, Vannevar Bush put forward the first vision of personal information management (PIM) system, Memex, by pointing out that the human mind "operates by associations" [5]. Hypertext systems, which flourished in the 80's [7], reinforced this vision and yielded the current World Wide Web. Recently, with the Semantic Web vision [3], a number of PIM systems associated with that vision, hence called *semantic desktop*, have been proposed, which support at least some of the following features:

Semantic data organization. Almost all existing approaches are trying to go beyond the hierarchical directory model. The critical factors of semantic data organization include adequate annotations, explicit semantics, meaningful associations, and a uniform representation [23].

Flexible data manipulation. A PIM system should enable the integration, exchange, navigation, and query processing of personal information in a flexible way. It should also be able to communicate (or interoperate) seamlessly with external sources (such as other PIM systems), e.g., in peer-to-peer (P2P) way [22].

Customizable rich visualization. As multiple visualizations can help the user understand pieces of data, a PIM system is supposed to support data visualization from different perspectives, e.g., the association-centric visualization [21] and the time-centric visualization [13, 12]. Ideally, users should be able to tailor such visualizations.

We have proposed a layered framework using multiple ontologies to organize personal information [24]. This layered framework enables a flexible and reusable system, by decoupling the domain and application ontologies, thus providing certain advantages over the use of a single domain model (e.g., [10]). In this paper, we describe in detail the architecture of our semantic desktop system, named MOSE (**M**ultiple **O**ntology based **S**emantic D**E**sktop) and the challenges that we are addressing in the course of its implementation.

Another contribution that we make in this paper is with respect to the *personal information application* (PIA) development, and hence to the inter-desktop information sharing and data integration by means of PIA-based *desktop services*. All these are motivated by the following example.

Example 1. A PhD student majoring in Chemistry has collected quite a few publications related to her research area and is now compiling a literature survey. The publications are stored as PDF files in different directories. For the literature survey she looks at a group of selected papers. For each of those papers, she would like to read some of the interesting papers that are referenced in that paper, which have already been downloaded and stored in the local desktop. To locate those papers, she can browse the directory hierarchy, use the search capacity provided by the operating system (if she can remember the file names), or use desktop search tools, such as the Google Desktop Search[1] or MSN Desktop Search.[2]

As the literature survey progresses, the student becomes tired of switching between windows, and wants to develop a bibliography management system such that the above

[1] http://desktop.google.com

[2] http://toolbar.msn.com

mentioned functionalities are integrated in a single interface. However, she finds it challenging to implement such a system, which requires several components, including a database to store and retrieve the citation relationships between pairs of publications. She asks the help of a friend majoring in Computer Science, who develops for her such a standalone application in Java. Now, the student is able to browse through her publications and the citation network easily. However, she would like to share that application with her advisor and with the other students in the project but is not able to do that. Furthermore, she would have liked to be able to access the publications that the other group members have discovered and stored, but cannot do that either.

When all the papers have been discovered and interrelated she would finally like to integrate the bibliography application with an application for paper composition. The paper composition application would gather several pieces of information such as related literature, experimental results, and comments/corrections from the advisor. However, she discovers that the two applications do not interoperate and she has to manually "import" the information that is gathered by the bibliography application into the paper composition application.

There are several key considerations in the design of a PIA development tool. First, end users may either be ignorant of programming skills or be reluctant to write such programs in the context of organizing the information in their desktop. Therefore, the PIA development environment, if provided, should hide the programming details from the user. The second consideration has to do with the flexibility and expressiveness of the designer. Even though we do not expect to invent another programming language, there are some fundamental functionalities that we need to make available, such as data access, data presentation, and business logic. Finally, there is the need to share the information related to the same application (or task) between two end users, as well as the need to reuse and to interoperate among existing PIAs.

Based on its semantic data organization, MOSE provides a semantic tool for end users to develop PIAs—the *PIA designer*. In this paper, we describe how we exploit the MVC (Model-View-Controller) methodology [16] for the personal information development in the PIA designer, which addresses the issues above illustrated. In particular, we discuss how PIAs can be formalized as *desktop services* and how such services can facilitate the data interoperation and intergration across semantic desktops.

The rest of the paper is organized as follows. After describing related work in Section 2, we present the system architecture and its main components in Section 3. Section 4 presents in detail the PIA development using the MVC method and the implementation of the PIA designer. In Section 5, we discuss desktop interoperation as provided by desktop services, whose definition we propose. We also discuss two cases of desktop service execution in MOSE. Finally, we present our conclusions and directions for future work in Section 6.

2 Related Work

The term of *semantic desktop* was first coined by Decker and Frank, who also stated the need for a "networked semantic desktop" that is enabled by several key emerging technologies including: the Semantic Web, P2P computing, and online social networking

[9]. The state-of-the-art of semantic desktop has been comprehensively summarized by Sauermann [23].

Among the existing approaches to PIM in desktops, the Gnowsis project aims at a semantic desktop environment that supports P2P data management based on *desktop services* [22]. Similarly to MOSE, Gnowsis uses ontologies for expressing semantic associations and RDF for data modeling. SEMEX is another personal data integration framework that uses a fine-grained annotation based on schemas, similar to our ontology-based framework [10]. However, a single domain model is provided as the unified interface for all data access, while we propose a layered framework using multiple ontologies to organize personal information. Our framework enables a flexible and reusable system, by decoupling the domain and application ontologies. MyLifeBits [13], Haystack [21], and Placeless Documents [11] are three PIM systems that support annotations and collections. The concept of *collection* is essentially the same as the conceptualization (using ontologies) of resources in our framework.

Our previous work uses ontologies to organize personal information and supports query processing within and across personal information applications (PIAs) based on a query rewriting algorithm. This algorithm requires that both PIAs have their ontologies mapped to each other [24]. In this paper, we propose PIA communication through the composition of desktop services.

Existing interfaces provide a workspace for the end user to develop applications. Such applications have their own data model, data presentation, and control logic. Of such interfaces, Haystack's end user interface [2] is the closest to the PIA designer that we introduce in this paper. Both interfaces support parameterizable channels that are collections of items retrieved by executing the channels (essentially queries). Their channel parametrization is oriented to individual channels, while ours can take the input from other channels so that two channels (with their associated views) can interact. Furthermore, the definition of view (i.e., data visualization) in both systems is different. We define a *view* as a visual component associated with a channel, which can be, for example, a graph, a list, or a text fragment. In comparison, their data views are the results of the execution of channel in the form of text. Finally, both Haystack and our approach allow for the reuse of previously defined channels and views. However, in our system, the channels and views are bound to the *desktop services* defined in terms of PIAs, so that the reusability is naturally implemented by desktop service composition, whereas Haystack does not provide a way to compose distributed desktop services.

Other interfaces for personal data management are based on Wikis and include SemperWiki [19] and WikSAR [1]. However, they resemble a hypertext composer (or content manager) providing the user with a means to put pieces of information together as a Wiki page.

3 System Architecture

Figure 1 presents the architecture of MOSE (**M**ultiple **O**ntology based **S**emantic D**E**sktop). The following describes the primary components of the framework, including the components related to semantic data organization (on the server side) and those related to semantic data manipulation (on the client side).

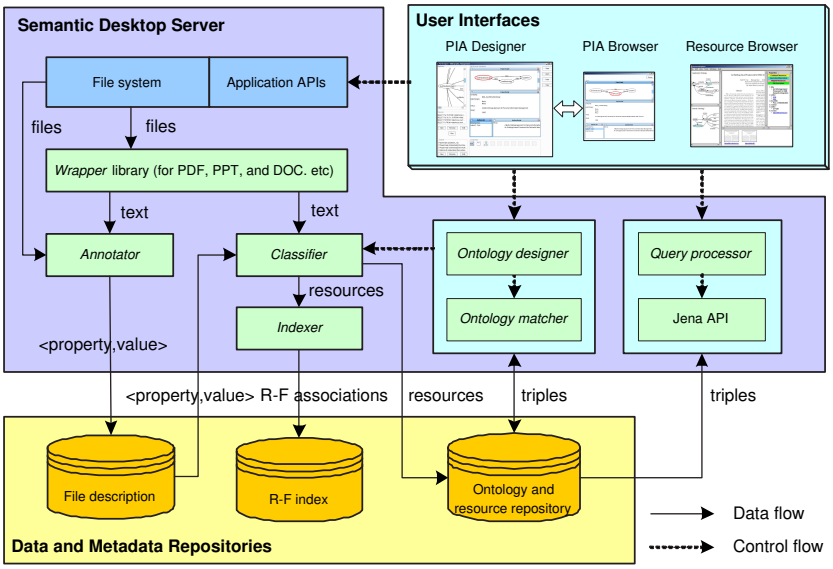

Fig. 1. The architecture of MOSE.

3.1 Semantic Data Organization

Our framework goes beyond the hierarchical directory based organization by means of two types of ontologies: domain ontologies and application ontologies. The former represent the conceptualization of different domains, thus providing a foundation for personal data classification. The latter are designed to serve as the data model underlying personal information applications (PIAs), which are developed by the end user. More details of how these ontologies cooperate to enable a semantically powerful data manipulation in the semantic desktop are given in Section 4.

File wrappers. The semantic organization is mainly based on a series of analysis and processing on text documents in the personal information space. That is, we do not consider the non-textual features of a file, although such features may facilitate data annotation [4]. A *file wrapper* is used to retrieve text from various types of files, such as PDF, PPT, and DOC. The other functionality of file wrappers is to obtain from the file system the system-defined properties of a file, e.g., its MIME type, size, and date.

Annotator. The annotator is responsible for creating and enhancing the annotation (or metadata) of a file. It is fed with the results of file wrappers, including the retrieved text and its standard properties, based on which it associates the file with property-value pairs. Most of current data annotators need input from users, although sometimes part of the annotations can be obtained from the file content. In practice, a semi-automatic annotator is often provided, such as the “easy” annotation mechanism of MyLifeBits [13]. In MOSE, the annotations are stored in a database, called *file description*.

Classifier. The classifier is one of the most important components for the semantic organization in the framework. Given a file and its file description, the classifier provides the following operations: (1) Identification of the file as a resource with a unique URI (Universal Resource Identifier); (2) Examination of the file content to explore the resources that are *contained* or *referred to* by the file; (3) Population of domain ontologies with all discovered resources; (4) Determination of the associations between resources, called *resource-resource (R-R) associations*. These resources and their associations are maintained in a *resource repository*.

Indexer. After being classified, a file is indexed in terms of the resources discovered in itself (e.g., the names of the authors in a publication). Such resource-file indices are stored in a repository, called *R-F index*, for the future use for query answering. There are three types of R-F indices (also called R-F associations): *identification*, *containment*, and *reference*, which are obtained by the first and second operations of the classifier. Given a query of keywords posed by the user, the query processor of MOSE can first locate the corresponding resources and then find the files that are identified as, containing, or referring to such resources, by means of the R-F index.

Ontology designer and matcher. At the center of the framework of MOSE are the multiple application and domain ontologies stored in the *ontology repository*. We provide an *ontology designer* for the management of concepts and roles of individual ontologies, and an *ontology matcher* for the maintenance of inter-ontology relationships (i.e., ontology mappings). Considering that most semantic desktop end users may lack the knowledge of particular ontology languages (e.g., RDFS or OWL), the ontology designer should hide the details of such languages but enable users to work with the conceptualization of their domains of interest. In addition, to improve the precision of an automatic ontology mapping process, the ontology matcher may be able to combine different ontology matching strategies [8, 15].

3.2 Semantic Data Manipulation

Based on the semantic data organization using multiple ontologies, MOSE provides the following interfaces for the user to manipulate personal information: a desktop-wide browser called *resource explorer*, and two other UIs, the *PIA designer* and *PIA browser*, for PIA development and execution, respectively. The functionalities of the PIA designer and browser are described in Section 4. The functionalities of the resource explorer are:

Context-aware browsing. The context of the data being browsed include the associated annotations, related domain ontologies, and other associated information (e.g., *person* and *time*).

Navigation by categorized associations. The ontology-based data classification and resource discovery establish various sorts of associations among the personal information items. The resouce explorer facilitates the viewing of such associations as organized in different categories. More details about the navigation functionality of the resource explorer, called *3D navigation*, can be found elsewhere [24].

Customizable visualization. The resource explorer provides a pane that shows the data that relates to the data currently being viewed. By default, the related instances are

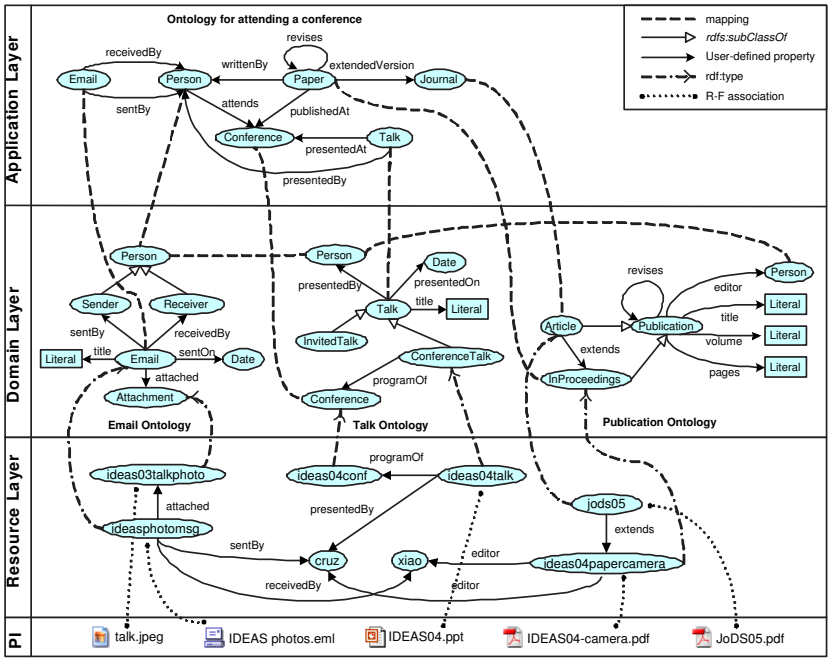

Fig. 2. An application ontologies is constructed for the PIA of conference attendance. The domain layer contains three ontologies for the domains of Email, Talk, and Publication, respectively. The resource layer presents all the triples (in graphs) representing the resource-file and resource-resource associations.

displayed in the form of the thumbnails, but other visualizations can be defined by the user.

4 Personal Information Applications

In this section, we first present the layered ontologies-based model, which lays a semantically rich foundation for the application design in MOSE. We then describe in detail the PIA development environment, the *PIA designer*, which supports flexible personal data management in MOSE.

4.1 The Layered Model with Multiple Ontologies

The following are the three layers of ontologies in MOSE, which act as a basis of the development of PIAs.

Resource layer. This layer comprises the low-level functions for maintaining the resources, the file description, the R-R associations, and the R-F index, thus acting as an intermediate layer connecting the personal data with the domain ontologies in the upper domain layer. The connections thus established result from the data classification process.

Domain layer. The domain ontologies in this layer have a double role. First, they are used as categorization of the resources in the resource layer. These domain ontologies are typically designed for different domains such as Conference, Person, Photo, and Email. There are a number of such ontologies available on the Web, for example, in the DAML Ontology Library,[3] the Semantic Web Ontologies,[4] or the OWL ontologies provided by the Protégé project.[5] Second, they serve as the basis for the construction of the application ontologies (in the application layer) through ontology mappings. In this sense, we say that the domain layer is *loosely coupled* with the application layer, thus providing flexibility and reusability.

Application layer. This layer contains the application ontologies, each of which may underly a PIA. As mentioned before, these application ontologies are defined as views over the domain ontologies. Different PIAs may have different application ontologies and are functionally independent from one another, since it is unlikely that a single ontology can cover various applications. However, PIAs (even if developed by different users in different PIM systems) can interoperate by means of mappings between their application ontologies, so as to integrate relevant information.

As a concrete example, Figure 2 shows a fragment of ontologies and instances in the layered model. We note that the ontologies and resources may be represented in different languages as long as they can be parsed by some available API. For example, Jena API, which is used by MOSE, is able to parse both RDFS and OWL (Web Ontology Language) ontologies.

4.2 MVC-based PIA Development

The resource explorer allows for the "global" exploration of the resources and ontologies in a desktop. However, views need to be tailorable for the users' diverse tasks, as we see in Example 1. To this end, MOSE provides a tool, the PIA designer, whose main objective is its flexibility.

Each PIA can work in a standalone mode, with its own application ontology, user interface, and work flows, aiming at a specific task (e.g., bibliography management, paper composing, or trip planning). Meanwhile, different PIAs can communicate with each other as in a P2P network, by means of the connections (mappings) established between their application ontologies. In MOSE, a PIA can present two modes: *development mode* and *execution mode*. The interfaces corresponding to these two modes are respectively the *PIA designer* (for the development mode) and the *PIA browser* (for the execution mode), which can switch from one to another at anytime.

The development of a PIA uses the MVC (Model-View-Controller) methodology. In particular, in the development of a PIA, the "Model" can be an application ontology that has been composed as a view over domain ontologies; the "View" consists of one or more components that present data in different forms such as graph, text, and list; the "Controller", which is the business logic of the PIA, is a set of "if-then" rules, which enable the interaction and synchronization between different data components.

[3] http://www.daml.org/ontologies/

[4] http://www.schemaweb.info

[5] http://protege.stanford.edu/plugins/owl/owl-library/

The data associated with components to be displayed are retrieved from the repositories of ontologies and instances by queries named *parameterized channels*.

The specifications of a PIA, as defined by the user by means of the PIA designer, including the model, view, and business logic, can be serialized in XML. It is called the PIA definition. Now, the user can run a PIA in the PIA browser, which interprets and executes the PIA in either an "online" mode (by directly switching from the designer to the browser) or an offline mode (by loading from the PIA's permanent serialization). The separation of the declarative specifications from the interpretative execution greatly benefits the communication between semantic desktops in terms of PIA interoperation, as we will see in the following sections.

4.3 Implementation

We have implemented a prototype of PIA designer using Java, as shown in Figure 3. Following the three basic elements of an application, the following describes three stages of the application development.

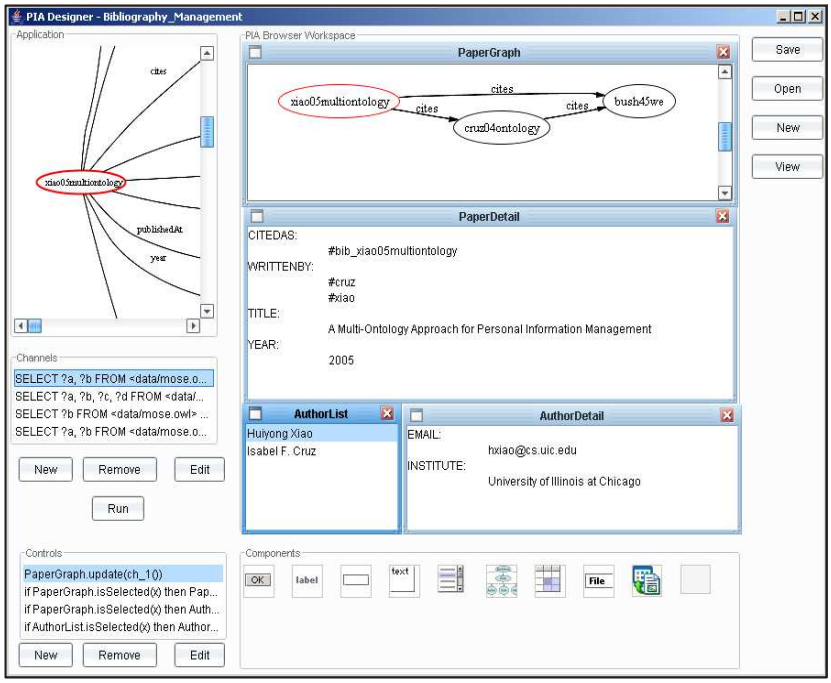

Fig. 3. The PIA designer.

Modeling. In the first stage, the user loads the application ontology from the ontology repository, which represents the model underlying the PIA to be designed; it will be

graphically shown in the *Data Model* pane. The application ontology is mapped to the domain ontologies, under which the resources representing personal information are classified. Actually, the application ontology is constructed as a view over the domain ontologies in a "global as view" (GaV) approach [17]. This mapping process should not require the users' programming expertise, but only their awareness of the task and their knowledge of the domain.

Visualization. The second stage involves the design of the layout of the PIA, with one or more visual components, each of which can be associated with a stream of data for its presentation. The user drags the desired visual components from the *Visual Component* pane to the *PIA Browser Workspace* pane. Examples of such components include TextPane, List, Table, Graph, and File. The associated data can be resources, strings, files, and whatever as instances of the ontologies; they are retrieved by queries, called *channels* (introduced in [21]), on the application ontology. Some components, such as Button, Label, TextInput, and MessageBox, are used to facilitate the interaction between the user and the PIA browser. A special component called Services is used for desktop service composition, as discussed in Section 5.

Controller and parameterized channels. In the final stage, the controller (or business logic) of a PIA is specified so as to realize rich interactions between the data and their views, and to synchronize several visualizations. These controllers manage all possible updates of the model and handle the events from the user interface, using "if-then" rules (more sophisticated controls will be considered in future work) of the following form:

if $\texttt{Component}_1\texttt{.event}_1(x_1)$ **and** ... **and** $\texttt{Component}_n\texttt{.event}_n(x_n)$
then $\texttt{Component}_1\texttt{.action}_1(y_1)$; ...; $\texttt{Component}_m\texttt{.action}_m(y_m)$;
endif

where x_i, $i \in [1..n]$, are parameters passed from the events, and y_i, $i \in [1..m]$, are the *channels* that result in the actions. It often happens that the response of a component to some event needs to take x_i as a parameter to execute y_i, especially when updating the data that is sensitive to x_i in a visual component. For this purpose, we introduce the concept of *parameterized channel*, which are channels that have their contents determined by the parameters at runtime. In MOSE, where channels are queries over ontologies, the parameter of a channel can be bound to a variable or a constant in the query. By means of parameterized channels, an event started from a component can pass any values to another component, thus enabling interactions between different components.

Example 2. As shown in Figure 3, at the top left corner, the user loads the application ontology (for publications), to develop a PIA for bibliography management. The application's user interface uses a Graph for displaying the citation network of papers, a TextPane for the paper's details, a List for the paper's authors, and a TextPane for the author's details.

To associate data with their proper visualization, the user defines the following channels, in the syntax of RDQL (RDF Data Query Language), which has an SQL-like grammar [14]. Each channel is in the form of string, which can then be fed into an RDQL interpreter (e.g., provided by the Jena API) for execution.

1. **ch_1()**: *"SELECT ?a, ?b WHERE (?a, cites, ?b)"*
2. **ch_2(**x**)**: *"SELECT ?a, ?b, ?c, ?d WHERE (" + x + ", title, ?a), (" + x + ", writtenBy, ?b), (" + x + ", year, ?c), (" + x + ", citedAs, ?d)"*
3. **ch_3(**x**)**: *"SELECT ?a WHERE (" + x + ", writtenBy, ?b), (?b, name, ?a)"*
4. **ch_4(**x**)**: *"SELECT ?a, ?b WHERE (" + x + ", institute, ?a), (" + x + ", email, ?b)"*

As an example of parameterized channel, the second query, $ch\_2(x)$, returns the title, author, year, and citation entry of a publication, which is bound to parameter x.

The data computed by executing a channel will present different forms depending on what visual component is used to visualize this data. For example, a Graph shows the data represented as a graph, where nodes are resources and edges are their associations. To construct such a graph, the nodes representing the same resource will be merged into a single one.

The following rules are defined to specify the controller, in which the first rule has no preconditions, thus being triggered at the very beginning of the PIA's run.

1. *PaperGraph.update(ch_1())*
2. ***if** PaperGraph.isSelected(*x*) **then** PaperDetail.update (ch_2(*x*))*
3. ***if** PaperGraph.isSelected(*x*) **then** AuthorList.update (ch_3(*x*))*
4. ***if** AuthorList.isSelected(*x*) **then** AuthorDetail.update (ch_4(*x*))*

5 Services-based Desktop Interoperation

As mentioned before, two PIAs can communicate in a P2P fashion based on the application ontology mappings established between them. It is required in this case that the two PIAs are designed for a similar task, for which they have their application ontologies partially or fully overlapping. We say that this way of PIA interoperation (or integration) is on the semantic level and is oriented to data models. Previous work has discussed such P2P ontology-based query processing [6, 25, 26]. In this section we discuss another type of PIA interoperation, which is realized by means of desktop services, thus called service-oriented interoperation.

The notion of *desktop service* was first introduced into the vision of semantic desktop by the Gnowsis system [22]. However, to our best knowledge, there has been no definition and formalization of desktop services. Next we give our own definition of what constitutes a desktop service in terms of parameterized channels, and describe how this service-based mechanism facilitates the data interoperation and integration in our semantic desktop vision. We assume PIA-based desktop services in our discussion, and use both terms, PIA and desktop service, interchangeably.

In general, a service (e.g., Web service[6]) must have its interface (i.e., input and output) defined, while keeping the implementation of its operation hidden from the service consumer. Intuitively, a PIA in MOSE consists of a set of visual components bound to parameterized channels. In this sense, we can see a channel as the minimal unit of service, taking the parameters as input and its resulting data as output. Starting

[6] http://www.w3.org/2002/ws/

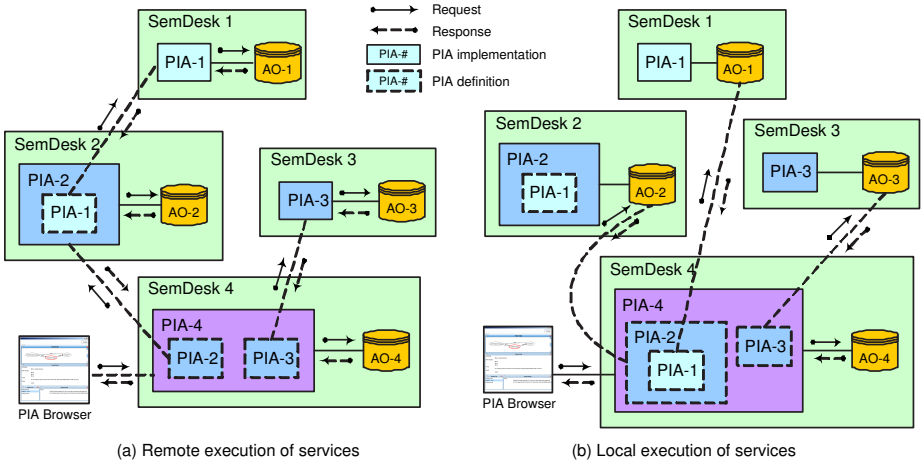

Fig. 4. Desktop services composition and execution.

from this point, we are able to give a definition of service based on the definition of parameterized channel.

Formally speaking, a *parameterized channel* q is a triple $\langle M, I, O \rangle$, where M is the underlying model (i.e., application ontology), I is a set of parameters (i.e., input), O is the set of tuples resulted from execution of the channel (i.e., output). A *desktop service* s is a 5-tuple $\langle Q, I, O, V, C \rangle$, where

- $Q = \{q_1, ..., q_m, s_1, ...s_n\}$, is a set of channels $q_1, ..., q_m$ or services $s_1, ...s_n$, where $m \geq 0$, $n \geq 0$, $m + n \geq 1$, and $s_i \neq s$, $i \in [1..n]$;
- $I \subseteq I_1 \cup ... \cup I_m \cup I'_1 \cup ... \cup I'_n$ is the input, where I_i is the input of q_i, $i \in [1..m]$, and I'_i is the input of s_i, $i \in [1..n]$.
- $O \subseteq O_1 \cup ... \cup O_m \cup O'_1 \cup ... \cup O'_n$ is the output, where O_i is the output of q_i, $i \in [1..m]$, and O'_i is the output of s_i, $i \in [1..n]$.
- $V = \{v_1, ..., v_l\}$ is the set of visual components, with v_i being the component of o_i, where $o_i \in O$, $i \in [1..l]$.
- $C = \{c_1, ..., c_k\}$ is a set of rules representing the control flows among the components.

The above recursive definition, based on the units of channels, allows for a flexible composition of desktop services. Besides its self-defined channels q_i, $i \in [1..m]$, a PIA can reuse any services s_i, $i \in [1..n]$, and embed them in itself, by establishing which channels o_j of s_i to be shown in which view v_j, $j \in [1..l]$. Then, the controller C consisting of if-then rules is used to specify the composition (control and data flows) among these channels or services in the PIA. Because of space limitations, we do not elaborate on the different types of service composition (e.g., "sequential" and "parallel" flows) [20]. Instead, we describe next how the service-oriented inter-desktop communication is implemented, by means of service composition and execution, in the two cases that are depicted in Figure 4.

The first case, as shown in Figure 4(a), is called *remote execution* of desktop services. In the example, there are four services (PIA-1 to PIA-4), with their respective application ontologies (AO-1 to AO-4). Suppose that PIA-4 is the starting point of the service execution, where the user interacts with the PIA browser. All requests for both the data and the execution of other services (defined and implemented in other desktops, but composed by the current service) are driven by events from such interactions. Whenever a nested remote service (e.g., PIA-2 or PIA-3) is triggered by the current service, a request for execution will be sent to the remote desktop (e.g., SemDesk 2 or SemDesk 3), where the remote service will be executed. As a response to the request, the remote service returns its execution results to the current service.

While the first case is similar to what happens with Web services, the second case of desktop service execution (called *local execution*, as shown in Figure 4(b)) is quite different. In particular, whenever a service nested in the current service is activated, it will be locally interpreted and executed by the PIA browser in the current desktop. However, the local execution of a remote service (e.g., PIA-2) needs permission to access relevant data (e.g., AO-2) from a remote desktop. If so, the data is then duplicated in the local desktop via a secure data transfer.

We note that the essential difference between the two cases of desktop service execution is related to a tradeoff between control permission and data access. This flexibility is important in a semantic desktop setting. Depending on their available resources, some desktops may be reluctant to take a heavy workload while some others may be concerned with the privacy of their data. Therefore, a desktop (when acting as a server) can choose whether to contribute its computing power or share its data.

6 Conclusions and Future Work

In this paper, we show how the multi-layered and ontology-based architecture of our semantic desktop, MOSE, enables a semantically rich environment for personal information management. We also stress the importance of a flexible and reusable system as supported by decoupling the domain and application ontologies. In particular, we described an MVC-based approach for personal information application (PIA) development in MOSE. We have formalized the concept of desktop service, building on the notion of parameterized channel, as proposed in this paper. Furthermore, we discussed how desktop services can facilitate data interoperation and integration across distributed semantic desktops.

While the envisioned semantic desktop can be seen as a miniature of the prospective semantic web, it has its particular features as well as challenges, such as automatic classification of personal information into ontologies, context-aware information search, and flexible tools for data manipulation and application development. In the future, we will work along the following two directions: (1) We would like to make the outlined functionality accessible to most end users, for example, by allowing natural language specifications to automatically formulate channels. In this context, the previous work on conversion of natural language questions to formal queries is of great interests [18]. (2) We will also work on mechanisms for defining, publishing, discovering, and composing desktop services so as to extend their current capabilities. Our

goal is to provide a semantic platform, where Web services and desktop services can be semantically integrated in a seamlessly way, so as to achieve data integration and application interoperability across semantic desktops.

References

1. D. Aumueller and S. Auer. Towards a Semantic Wiki Experience – Desktop Integration and Interactivity in WikSAR. In *Proc. of the 1st ISWC Workshop on The Semantic Desktop*, 2005.
2. K. Bakshi and D. R. Karger. End-User Application Development for the Semantic Web. In *Proc. of the 1st ISWC Workshop on The Semantic Desktop*, pages 123–137, 2005.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
4. S. Bloehdorn, K. Petridis, C. Saathoff, N. Simou, V. Tzouvaras, Y. S. Avrithis, S. Handschuh, I. Kompatsiaris, S. Staab, and M. G. Strintzis. Semantic Annotation of Images and Videos for Multimedia Analysis. In *ESWC 2005*, pages 592–607, 2005.
5. V. Bush. As We May Think. *The Atlantic Monthly*, 176(1):101–108, 1945.
6. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. What to Ask to a Peer: Ontolgoy-based Query Reformulation. In *Proc. of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 469–478, 2004.
7. J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, 1987.
8. I. F. Cruz, W. Sunna, and A. Chaudhry. Semi-Automatic Ontology Alignment for Geospatial Data Integration. In *Proc. of the 3rd Int. Conf. on GIScience*, pages 51–66, 2004.
9. S. Decker and M. Frank. The Social Semantic Desktop. In *Proc. of the WWW Workshop Application Design, Development and Implementation Issues in the Semantic Web*, 2004.
10. X. Dong and A. Y. Halevy. A Platform for Personal Information Management and Integration. In *CIDR 2005*, pages 119–130, 2005.
11. P. Dourish, W. K. Edwards, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. B. Terry, and J. Thornton. Extending Document Management Systems with User-specific Active Properties. *ACM Transaction of Information System*, 18(2):140–170, 2000.
12. E. Freeman and D. Gelernter. Lifestreams: A Storage Model for Personal Data. *SIGMOD Record*, 25(1):80–86, 1996.
13. J. Gemmell, G. Bell, R. Lueder, S. M. Drucker, and C. Wong. MyLifeBits: Fulfilling the Memex Vision. In *ACM Multimedia 2002*, pages 235–238, 2002.
14. HP Labs. RDQL - RDF Data Query Language. http://www.hpl.hp.com/semweb/rdql.htm, 2005.
15. Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: the State of the Art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.
16. G. E. Kramer and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August/September 1988.
17. M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS 2002*, pages 233–246, Madison, Wisconsin, June 2002. ACM.
18. Y. Li, H. Yang, and H. V. Jagadish. NaLIX: an Interactive Natural Language Interface for Querying XML. In *SIGMOD 2005 (Poster)*.
19. E. Oren. SemperWiki: a Semantic Personal Wiki. In *Proc. of the 1st ISWC Workshop on The Semantic Desktop*, 2005.
20. C. Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, 2003.

21. D. Quan, D. Huynh, and D. R. Karger. Haystack: A Platform for Authoring End User Semantic Web Applications. In *ISWC 2003*, pages 738–753, 2003.
22. L. Sauermann. The Gnowsis Semantic Desktop for Information Integration. In *WM 2005*, pages 39–42, 2005.
23. L. Sauermann, A. Bernardi, and A. Dengel. Overview and Outlook on the Semantic Desktop. In *Proc. of the 1st ISWC Workshop on The Semantic Desktop*, 2005.
24. H. Xiao and I. F. Cruz. A Multi-Ontology Approach for Personal Information Management. In *Proc. of the 1st ISWC Workshop on The Semantic Desktop*, pages 19–33, 2005.
25. H. Xiao and I. F. Cruz. Integrating and Exchanging XML Data Using Ontologies. *LNCS Journal on Data Semantics, Springer Verlag*, 2006. (To appear).
26. H. Xiao and I. F. Cruz. Ontology-based Query Rewriting in Peer-to-Peer Networks. In *Proc. of the 2nd International Conference on Knowledge Engineering and Decision Support*, 2006.