On Derived Data Services in Cyberspace

Shi-Cho Cha
Department of Information Management
National Taiwan University
Taipei, Taiwan
csc@mba.ntu.edu.tw

Yuh-Jzer Joung
Department of Information Management
National Taiwan University
Taipei, Taiwan
joung@ntu.edu.tw

Abstract

We propose a framework, called Derived Data Services (DDS), to ease the tension between (a) the need to mine individual's Web usage logs across multiple sites for aiding in personalization, and (b) the inherent privacy risks in it. In DDS, a standardized, hierarchical, format can be developed, whereby some higher level abstractions from a usage log can be captured, and potentially used across multiple applications. The hierarchical data structure also allows derived data to be processed incrementally based on the level of information needed, and on the level of privacy an individual wishes to have. The derived data summarizes an individual's profile in cyberspace, and should be the legal property of the individual so that access to the profile must be legally authorized by the person.

1 Introduction

Customization is an important feature for information services and applications. Customization allows each user to get the best possible service out of the applications according to his/her needs and preferences. Service providers can also improve their customers' loyalty and lifetime value through customization because better quality of service can be provided. To understand customers' needs and preferences, service providers may collect and use their behaviors logs. For example, association rules can be discovered from behavior logs so that personal recommendations can be provided [1, 18]. Web sites can also adapt themselves based on users' navigation patterns to provide a more efficient way for browsing [28].

Because of the importance of behavior logs, many service providers treat user behavior logs as assets, and use them to find their niche over their competitors. Likewise, they usually maintain their users' behavior logs in their own databases independently. However, very few service providers can offer all types, or even a broad category, of services. Even for a specific type of services (e.g., online shopping), there may be several competitors, and a user may shop between them to find the best deal. So most service providers have only partial knowledge about their users. Consequently, they may incorrectly interpret their users using the partial and incomplete information they collected [27]. For example, when a person only buys books about computer science in an online bookstore, it may not imply that he is only interested in this field. If we use this information to filter out messages he is not 'interested' in, some useful information to him may be dropped.

Intuitively, if a person's behavior logs in different services and applications can be shared, then services and applications can project the person from a broader aspect, and so better quality of service can be provided. Furthermore, if a person can bring his personal data from one service to another, then the switching cost between different services can be considerably reduced. This is because, otherwise, a new service would have to learn about the user all over again. It may then take some time (perhaps after a number of visits by the user) for the new service to offer a satisfactory customized service to the user.

Several architectures have been proposed to enable a customized service and application to use a person's behavior logs in other services and applications. For example, *Global Customization Engine* is proposed to enable one application service to 'subscribe' a user's logs of another under the latter's consent [2]. The mechanisms of [12, 11] let users collect their click-stream data and store them into a centralized place so that another Web site can obtain users' click-streams of other sites from this place.

The sharing of behavior logs raises two problems, which have not been fully addressed in the literature. The first one is to understand the 'meaning' of a log: An application cannot utilize a log if it cannot understand the log. For example, when a person buys '*Overcoming Depression and Manic Depression (Bipolar Disorder), A Whole-Person Approach*' in an online bookstore, the action may be logged as a URL of 'http://bookstore/buy?bookid=BK991234689'. Such a URL log hardly conveys any useful information to other sites. The second problem, perhaps the most important one, is *privacy*. Although the sharing of behavior logs does bring some benefit in customized services, one is hardly convinced and comfortable to reveal every detail of his behavior logs in cyberspace!

To address the above two problems, we propose a framework called *Derived Data Services* (DDS). DDS provides a hierarchical architecture for using metadata to describe user behaviors in (Web-based) services. The architecture aims to provide an open and standard environment to describe behavior logs. This feature then allows logs in different contexts to be integrated together, and also allows new attributes to be added to the logs should applications need.

The metadata also embed some semantics of the logs so that logs generated in one context can be understood in another. The hierarchical nature of the architecture allows users to set their privacy level by determining how many layers of information they wish to reveal.

To allow user logs to be shared while protecting privacy, a service that wishes to use the logs (and their metadata) must provide a *derived data generation proposal* to DDS. The proposal specifies what data are to be derived from the logs, and how to derive the data. If the user accepts the proposal, DDS will generate the derived data based on the proposal, and then allows the service to access the data. In this regard, DDS can be viewed as an abstraction mechanism that provides services with high level descriptions of a user's characteristics, but hides the detailed logs of the user's behaviors.

The rest of the paper is organized as follows. Section 2 surveys related work on logs integration. Section 3 gives an overview of the framework. Sections 4-6 discuss key components in the framework. Conclusions and future work are offered in Section 7.

2 Related Work about Logs Integration

A person's behavior logs in different services can be obtained and integrated in several ways. First, the log data may be obtained from the services the person has visited. Observe that Web sites often log user requests. To allow their logs to be shared, heterogeneity between them must be solved. For this, technologies of federated database systems can be used. By viewing the data storage of each Web site as a component database of a federated database system, data in different component databases can be joined through tightly coupling or loosely coupling approach depending on how the component databases are integrated [30]. In the *tightly coupling* approach, one or more shared schemas are needed to be reconciled in advance. Log data of different sources are then integrated into the schemas and can be queried by users. However, it is very hard to reach an agreement among Web site administrators in the schemas, especially when new services may emerge everyday.

In the *loosely coupling* approach, each service reveals the schema of its log data. In contrast to the tightly coupling approach, each requester now interacts with each component database directly. A requester first decides which component databases he would like to access based on the disclosed schema of each component database. The requester can then use a multi-database manipulation language, such as MDSL [23], to define his own schema, to specify how to map data from schemas in different databases to the user's schema, and then to query data from the selected databases. Although the loosely coupling approach need not negotiate with each service to achieve an agreement about shared schemas, requesters must follow the schema of each service. Some request cannot be satisfied if the requested data is not included in the schema of the corresponding service. For example, information about the total amount of a commodity in a transaction cannot be obtained if it is not included in the log.

On the other hand, a centralized mechanism can also be used to track a person's behavior.[1] The centralized mechanism can be implemented in the following ways:

- The tracking mechanism can be implemented as a common component in each service. For example, *Universal Profile Services* such as Microsoft's Passport [25] or AOL's Magic Carpet [26] can be viewed as a kind of Single Sign-on service that allows a user to access different information services with only a single action of authentication and authorization. The tracking mechanism can then be incorporated into a Universal Profile Service so that user behaviors in using different services through the Universal Profile Service can be collected. Furthermore, an online advertisement company, such as DoubleClick [16], may collect a person's browsing behaviors in different sites while the person is viewing the company's advertisements in these sites.

- The tracking mechanism can be implemented as an agent in a person's computer to represent the person to interact with the site offering services so that the person's behaviors can be logged transparently. This kind of technology is widely used in, e.g., *personalized search* [9], *information filtering* [31], and *personalized recommendations* [19, 28]. The logs may also be requested by a service. For example, [12, 11] propose an architecture for a Web site to ask a user's agent through P3P (Platform for Privacy Preference) [14] and XML-QL [15] to access the XML formatted log data of the user.

Our DDS also adopts a tracking mechanism to track user behaviors. The reason for this design choice is that the Web is quite *dynamic* (because new services and new requirements of log data may emerge everyday) and *competitive* (because service providers usually compete with one another). The behavior tracking mechanism does not need to obtain an agreement among service providers, nor does it require them to modify their systems.

The next question is how to represent behaviors into log data. If the mechanism only records the URL of a user's HTTP requests, then no matter what kind of actions a person may take (e.g., searching with some keywords, browsing a document, adding a product into a shopping cart, etc.), the tracking mechanism may get only a URL string of the person's request. As mentioned before, such a URL usually does not convey enough information for further process. So the tracking mechanism should also know how to relate a URL to a specific action. Because we wish the tracking mechanism to work transparently between the user and services, an automatic way to let the mechanism know what actions the user is taking is needed. For this, DDS uses an *Annotator* module to extract raw data (including users' requests and the responses from services), and translates them into an 'understandable' format. The details will be introduced in the following sections.

[1] Note that 'centralization' here refers to the administration, not necessarily to the implementation.

3 System Overview

Generally speaking, DDS tracks user behaviors in different services, describes them in an extensible data model, and provides queries to the patterns through some derived data generated from the logs. Because logs or even derived data are also part of personal data (that contain other type of data such as name, gender, birthday, blood type, job title, etc.), DDS is designed to be integrated into a profile management service. The profile management service may be operated by a single service provider (like Microsoft's Passport), implemented as a component of a user's Web browser, deployed in a peer-to-peer network [7], etc. Therefore, some common components between DDS and the profile management service, such as authentication and security management, are assumed to be provided by the profile management service, and so will not be addressed in this paper.

The architecture of DDS is depicted in Figure 1. First of all, requests to services and responses from them are tracked by a behavior tracking mechanism as described in the previous section. The tracked data are then transmitted to the DDS kernel. Because it is impractical and often resource-wasting to store the whole tracked data into DDS, useful information needs to be extracted from the data to represent the behaviors. For this, the following two issues need to be addressed:

- The first problem is what information needs to be extracted or retained from the tracked data? Observe that the amount of data need to represent behavior logs depends on the amount of information one wishes to obtain from the logs, and certainly depends as well on the type of behaviors. Nevertheless, there are some common and basic features among different behavior logs, e.g., time, URL, and the engaging parties of an event. These features can therefore be represented as base log entries, on top of which different classes of metadata can be defined as additional information to describe different types of behaviors. This will be discussed in Section 4.

- To handle the heterogeneity among services, different *application-specific annotators* can be developed. Each annotator takes responsibilities of converting tracked data of the corresponding services into behavior logs of DDS. The annotators will be discussed in Section 5.

Finally, as mentioned in Section 1, DDS can use behavior logs to generate derived data. For this, DDS provides an architecture for service providers and experts to define derived data with XQuery [4] language in a *derived data generation proposal*. We will introduce the components of a derived data generation proposal in Section 6. Then, the *Derived Data Manager* generates derived data from an agreed proposal. As such, service providers can understand a person's behavior from generated derived data; while from the user's perspective, he can benefit from the sharing of his behavior patterns (e.g., improved quality of service) without revealing details of his behavior logs.

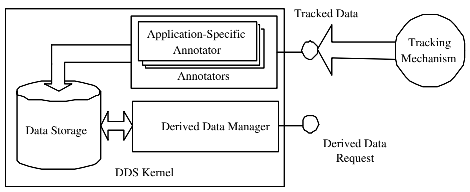

Figure 1. The Derived Data Service architecture.

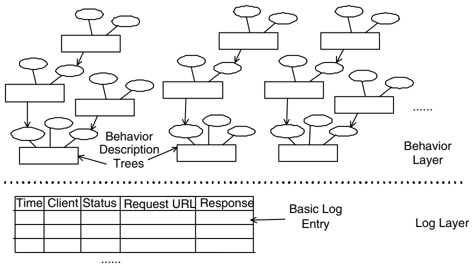

Figure 2. Logical view of behavior logs.

4 The Behavior Logs

We introduce behavior logs of DDS from a logical point of view, as well as from a physical point of view in this section. Logically, behavior logs are organized as in Figure 2. It consists of two layers: The *log layer* contains basic log entries that record a user actions (requests and responses) in a service. The *behavior layer* contains *behavior description trees* that abstract information derived from the actions. There are several types of behavior description trees, each of which represents a kind of behavior, e.g., online purchase, Web browsing, search, etc. A behavior may involve a sequence of actions.

The ontology of behavior description trees is shown in Figure 3. It specifies what *metadata* (represented by rectangles) and *properties* (ovals) should be used to describe a behavior description tree. The root node of a behavior description tree represents a behavior. It is associated with some properties describing basic information of the behavior, such as behavior type, ID, time, and places. Additional metadata are attached to give further description about the metadata if needed. For example, two behavior specific metadata are defined in the figure: one for Web browsing, and the other for online purchase. For Web browsing behavior, a metadata of type 'Page Description' is used to describe the Web page being browsed, such as keyword, subject, author, title, etc. For online purchase, a metadata 'Product Description' is used to describe the product, and a metadata 'Shipment Description' is used to describe the shipping. In general, if a metadata needs further information to describe itself, the information is described by another metadata in the layer above. This hierarchical organization has the following two important advantages:

- Extensibility can be achieved easily by defining a new (or a new version of) metadata when a new kind of behavior patterns or data requirement emerges.
- Because metadata can be divided by layers, data can be requested and processed incrementally based on the level of information needed, and on the level of privacy one wishes to have.

In addition to the metadata, several domains of categories are defined and used to represent the semantics of some specific data. The concept of domain categories is borrowed from philosophy to provide knowledge about an interest domain. For example, there are three kinds of domains in Figure 3: one for Web pages, one for products, and one for geographical locations. The hierarchical structure of a domain again allows one to apply different access control/privacy level to different categories.

For example, suppose a user has purchased a book '*Overcoming Depression and Manic Depression (Bipolar Disorder), A Whole-Person Approach*' from an online bookstore. Then in the corresponding behavior description tree, the 'Product Category' element will record that the item belongs to the 'Depression' category in the 'products domain', and the 'Geographical Category' element will record the city of the user, say 'Taipei'. Given the domain categories for products, we can easily see that the item belongs to 'Mental Health' of the 'Books' category; and given the domain categories for geographical locations, we can see that the user is in 'Taiwan', 'Asia'. So when a derived data is defined to calculate the sum of online shopping expenses grouped by categories, depending on how much information the user wishes to reveal to a site, some site may know that the person has spent \$25 on a book about Depression at a specific time and place; others may only know that a person in Asia has recently spent \$25 for a book.

From the physical point of view, the above data hierarchy can be implemented in several ways. We first observe that existing log files usually follow Common Log Format [24] or a similar but proprietary format. One disadvantage of using Common Log Format is the lack of a well-known language and tool for data query and request. In contrast, data stored in a relational database management system (RDBMS) can be queried through SQL. However, because we wish DDS to be implemented as a component of user's Web browsers, platform-independency becomes an important consideration. If data need to be physically stored in RDBMSs, then each user may need to install an RDBMS in his computer. This may then complicate the deployment of DDS. Therefore, XML is used to resolve this problem.

In DDS, each base log entry is represented as an XML element of *LogEntry*. The schema of LogEntry is shown in Figure 4. It is based on the fields used in the Common Log Format. Each LogEntry also has a *logID* attribute as its identity in each log file. Moreover, the *Referer* attribute can be used to represent the relationship between LogEntry elements. The LogEntry elements of a user are then collected into an XML document.

As described before, information derived from log entries is built into behavior description trees. To be able to cooperate with XML formatted log entries, the Resource

```
⟨?xml version='1.0'?⟩
⟨schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"⟩
⟨element name="LogEntry" type="BaseLogEntry" /⟩
⟨complexType name="BaseLogEntry"⟩
   ⟨attribute name="HostUsed" type="string" /⟩
   ⟨attribute name="Date" type="date" /⟩
   ⟨attribute name="Request" type="string" /⟩
   ⟨attribute name="Status" type="string" /⟩
   ⟨attribute name="Bytes" type="positiveInteger" /⟩
   ⟨attribute name="logID" type="ID" /⟩
   ⟨attribute name="Referer" type="IDREF" /⟩
⟨/complexType⟩
⟨/element⟩
⟨/schema⟩
```

Figure 4. Data schema of LogEntry.

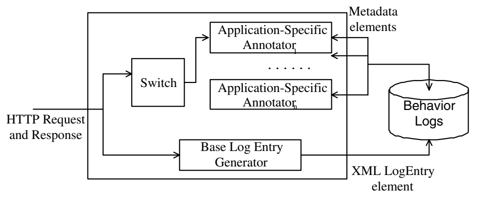
Metadata elements
Application-Specific Annotator_1
Switch
......
Application-Specific Annotator_n
Behavior Logs
HTTP Request and Response
Base Log Entry Generator
XML LogEntry element

Figure 5. The architecture of Annotators.

Description Framework (RDF) model [22] is used. As such, OWL Web Ontology Language [32], which is built on RDF Schema (RDFS) [5] and XML Schema [17], can be used to define ontologies of behavior data.

Domain categories are also represented as a set of classes. The concept of class inheritance is used here to represent the hierarchical relationship among categories. Examples for online purchase schemas, domain category schemas, and behavior description trees represented in RDF, can be found in [6].

5 Log Annotation

Behavior description trees described in the previous section are generated by application-dependent Annotators. The architecture of *Annotators* is depicted in Figure 5. The Annotator receives tracked HTTP requests and responses from a behavior tracking mechanism and converts them into metadata of the trees. More precisely, when an Annotator receives a pair of HTTP request and the corresponding response, the *Base Log Entry Generator* generates an XML LogEntry element based on the data pair and stores it into the user's base log file.

The *Switch* component then assigns the data pair to a corresponding application-specific annotator, which then uses the data pair to generate rdf:Description elements to describe the generated XML LogEntry element. For some applications, an annotator may need a sequence of data pairs to generate a complete behavior description tree.

At this point one may wonder who is going to provide the application-specific annotators? As described in Section 1, DDS-like services reduce the switching cost while a person is using a new service. To service providers, however, this

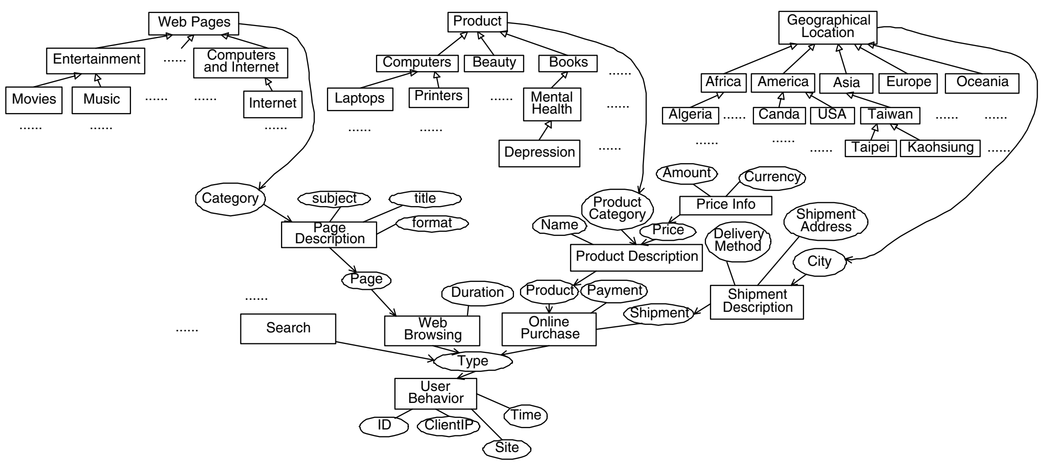

Figure 3. Ontology of a behavior description tree.

may reduce their competition because they may not be able to differentiate with one another by accumulating interaction experiences from their users. So they may be reluctant to support such a service (i.e., to provide enough information about the parameters used in their services). Therefore, several general annotators have to be developed, say, by some consortium, to deal with common user behaviors.

For example, a Web page may be described by, in addition to the keywords in the page, a category it belongs (see the Web page domain categories in Figure 3). For this, Yahoo provides a huge and exhaustive hierarchy of categories covering almost every aspect of Web documents. Automatic document processing techniques have also been developed to classify Web documents into the Yahoo categories hierarchy [21]. Moreover, technologies such as text summarization [3, 34] and keywords extraction [29] have also been developed to automatically extract features from a Web page. By applying these techniques, a general *Web browsing behavior* annotator can be implemented to generate behavior description trees for Web browsing behavior [8]. This annotator can then be used as default if no matching application-specific annotator can be found.

When the number of DDS users increases, they can then influence service providers to support such a service because otherwise they can turn into those service providers which have provided such support when choosing their service providers. Note that to support DDS, a Web site can simply add Dublin Core metadata [33] in its Web pages to describe the pages such as keyword, subject, and abstract. An application-specific annotator can then use this information directly to generate behavior logs for the browsing behavior. This greatly simplifies the design of annotators as well as eases the interface between DDS and service providers.

6 Derived Data Generation

DDS enables services to understand their customers through derived data without revealing their detailed logs and behaviors. For example, service providers may wish to know their customers' life-style (e.g., *A-I-O items*, which stands for the activities, interests and opinions) for marketing purposes [20]. In tradition, the data are collected through communication with the customers and behaviors observation. Through DDS, service providers can now obtain a user's lifestyle directly from the derived data released by the user.

Derived data can be generated dynamically from behavior logs upon request, or they can be precomputed and queried by service providers. In DDS, we have opted for the second approach for performance reasons. This is because derived data interested to service providers such as A-I-O items are often not very dynamic. For example, a person's purchase behavior would not vary dramatically within a short period of time. Therefore, on-the-fly generation of such data is not needed and can only slow down the performance of the system. Another reason is that derived data are part of personal data. So they can be integrated into a person's personal profile to which service providers can query via existing protocols such as P3P, X.500, or LDAP. As such, no extra interface or protocols are needed between the person and the service providers.

To generate derived data, DDS provides an interface to let experts (and service providers) to define the data via *derived data generation proposals*. Figure 7 gives an example of a derived data generation proposal that generates the distribution of a person's online purchase. In general, a proposal includes the following components:

- Each derived data should have a relatively distinguishable name for requesters to identify them. This is specified in the *Name* element.
- The *Query* element contains information about how the derived data is to be generated. Because both log entry elements and their descriptions are represented in XML, XQuery language is adopted.
- Because not everyone can understand XQuery queries and their effects, a text format description is included. The information is included in the *Consequence* element.

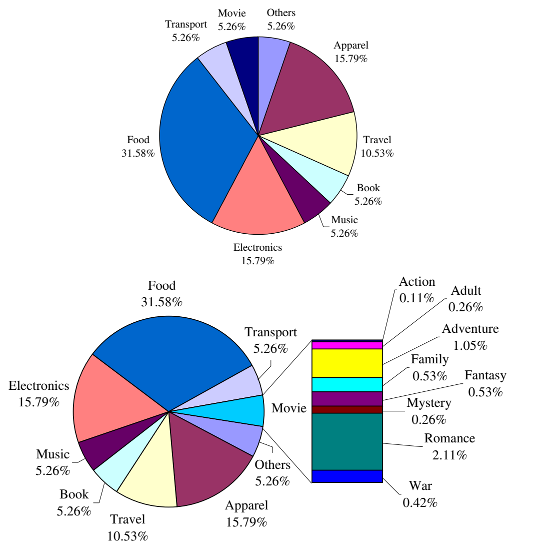

Figure 6. A person's online purchase distributions. The top shows only the first-level categories, while the bottom shows a more detailed distribution of the movie category.

- The *ProposedBy* element shows who defined the derived data. Because not every people has the ability to judge the effect of derived data when they are revealed, certification organizations may be needed. The *VerifyBy* element shows which certification organization has checked the proposal. Moreover, digital signature is used for verification.

- The *UpdateScheme* element tells the Derived Data Manager when the derived data need to be refreshed, e.g., they may be re-calculated periodically or every time a request arrives.

Recall that a person can decide how much information he wishes to reveal by determining how many layers of data a requester is allowed to access. Therefore, different requesters with different trust levels may obtain different values about the same derived data. For example, if a person allows everyone to view only the first-level categories in his online purchase logs, then the derived data 'Purchase Distribution' generated by the proposal in Figure 7 can contain only the first-level category distribution as that shown on the top of Figure 6. On the other hand, if the person allows video rental companies to access detailed categories of his purchase on movies, then they may be able to obtain a more detailed purchase distribution as shown on the bottom.

Recall that derived data usually need not be re-calculated for every request. So they can be cached to improve system performance. However, because to the same derived data, different requesters may have different views depending on their trust levels, traditional cache schemes may not work well here. For this, DDS provides a *template-based caching* architecture. As sketched in Figure 8, the Derived Data Manager contains the following components: The *Proposal*

```
⟨DerivedDataProposal⟩
⟨Name⟩Purchase Distribution⟨/Name⟩
⟨Consequence⟩ The total amount of online purchases will be
  generated and grouped by categories
⟨/Consequence⟩
⟨Query⟩
defi ne function summaryCategoryExpense($root_c as
    element(Category), $allcat as element(Category)*,
    $purchased as element(rdf:Description)*)
    as element(CategoryExpenseSummary){
  for $subcategory in $allcat
  where $allcat/rdfs:subClassOf/@resource = $root_c/@rdf:ID
  return {
    ⟨CategoryExpenseSummary⟩
      let $tempamount := double(0)
      for $catpurc in $purchased
      where $purchased/ProductCategory/type/@resource =
        $root_c/@rdf:ID
      return {
        $tempamount += double(CountExp($catpurc)/amount)
      }
      let $ce := summaryCategoryExpense($subcategory)
      $tempamount += double($ce/ExpenseSummary)
      {$ce}
      ⟨ExpenseSummary⟩$tempamount⟨/ExpenseSummary⟩
    ⟨/CategoryExpenseSummary⟩
  }
}
defi ne function recentRelated($alld as element(rdf:Description)*)
    as element(rdf:Description)* {
  for $behavior in $alld
  where $behavior/type/@rdf:resource = "#UserBehavior"
    and get-gMonth-from-dateTime($behavior/Time)
  return {
    {$behavior}
    getAllRelatedDescription($behavior)
  }
}
let $allcategories := doc(ontology.xml)//Category
let $productc := doc(ontology.xml)//Category[rdf:ID="000"]
let $desc := recentRelated(doc(behavior.xml)//rdf:Description)
return {
  let $ce := summaryCategoryExpense(productc, $allcategories, $desc)
  {$ce}
}
⟨/Query⟩
⟨ProposedBy⟩example issuer⟨/ProposedBy⟩
⟨VerifyBy entity="verify.org" algorithm="DSA"
  signature="MC0CFQCHapEh+cL14In5fYeyl580uj6t...cwIUP/ZVs"/⟩
⟨UpdateScheme type="periodically" period="86400" /⟩
⟨/DerivedDataProposal⟩
```

Figure 7. A derived data generation proposal.

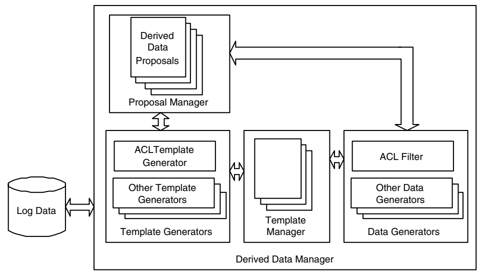

Figure 8. The architecture of the Derived Data Manager.

Manager manages the agreed proposals and provides necessary information for the process of derived data generation. The *Template Generators* generate templates based on the proposals. A *template* is similar to a multi-resolution cache object [10] and an XSL stylesheet [13]. It contains information for the *Data Generators* to generate different 'views' based on different permissions to the levels of behavior logs.

For example, a simple *ACL Template Generator* can be implemented by attaching ACLs (Access Control Lists) to raw data (on which derived data depend). In addition, an *ACL Filter* can be developed to filter out unauthorized components from the template produced by the ACL Template Generator. To illustrate, suppose a derived data (V_1, V_2) depends on the raw data V_3, V_4, and V_5, where $V_1 = V_3 + V_4$, and $V_2 = c \times V5$. When a template about the pair is generated, the ACL Template Generator attaches the ACLs of V_3 and V_4 to V_1, and the ACL of V_5 to V_2. If a requester wishes to obtain the pair, but he has no permission to view V_3, then V_1 will be filtered out and only V_2 is provided.

For some applications, rather than filtering out entirely a component of a derived data, a more sophisticated template may be developed to leave the component, but removing the effect of the raw data on which the component depends but a requester has no permission to view it. This will be left to our future work.

7 Future Work

Other than a concrete implementation of DDS, there are many interesting things left to be done. First, we observe that this paper focuses on user behaviors in using services in cyberspace. Similar methods may be used to develop a framework for studying user behaviors in more general applications. Secondly, current design in DDS does not consider storage limitation. The retention of outdated logs and their destruction time may be considered to save storage space. Third, XQuery is used in DDS for derived data definition. However, XQuery provides only simple aggregate functions, such as sum, count, min, max, etc. More complicated data mining functions, such as finding association rules or sequence patterns, may be included. Finally, in case a user wishes to manually modify the generated derived data, should such modification be allowed? If so, should requesters be aware of such modification, and how?

References

[1] G. Adomavicius and A. Tuzhilin. Using data mining methods to build customer profi les. *Computer*, 34(2):74–82, Feb. 2001.

[2] R. M. Arlein, B. Jai, M. Jakobsson, F. Monrose, and M. K. Reiter. Privacy—preserving global customization. In *Proc. ACM EC*, pp. 176–184, 2000.

[3] A. L. Berger and V. O. Mittal. Ocelot: a system for summarizing web pages. In *Proc. ACM SIGIR*, pp. 144–151. 2000.

[4] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery 1.0: An XML Query Language. In *W3C Working Draft*, 2002.

[5] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. In *W3C Working Draft*, 2003.

[6] S.-C. Cha. *On the Realization of a Universal Profi le System in Cyberspace*. PhD thesis, National Taiwan University, Taipei, Taiwan, 2003.

[7] S.-C. Cha, Y.-J. Joung, and Y.-E. Lue. Building universal profi le systems over a peer-to-peer network. In *Proc. WIAPP*, pp. 142–151, 2003.

[8] S.-C. Cha, W. Yang, and Y.-Z. Joung. The integration of web browsing behavior. In *Proc. 4th Taiwan Conference on Doctoral Consortium in Information Management*, pp. 7–12, Shanghai, China, Apr 2002.

[9] L. Chen and K. Sycara. WebMate: A personal agent for browsing and searching. In *Proc. Agents*, pages pp. 132–139, 1998.

[10] J. H. Chim, R. W. Lau, and A. Si. Multi-resolution cache management in digital virtual library. In *Advances in Digital Libraries Conference*, pp. 66–75, Apr 1998.

[11] I. Cingil. Supporting global user profi les through trusted authorities. In *ACM SIGMOD Record*, volume 31, pp. 11–17, 2002.

[12] I. Cingil, A. Dogac, and A. Azgin. A broader approach to personalization. *CACM*, 43(8):136–141, Aug. 2000.

[13] J. Clark. XSL Transformations (XSLT), Version 1.0 W3C Recommendation. `http://www.w3.org/TR/xslt`, Nov. 19, 1999.

[14] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. Platform for Privacy Preference (P3P). In *W3C Recommendations*, 2002.

[15] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *Submission to the World Wide Web Consortium*, 1998.

[16] DoubleClick Inc. http://www.doubleclick.com/.

[17] D. C. Fallside. XML Schema Part 0: Primer. In *W3C Recommendation*, 2001.

[18] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In H. Liebermann, editor, in *Proc. IUI*, pp. 106–112, Jan. 9–12 2000.

[19] T. Joachims, D. Freitag, and T. M. Mitchell. Web watcher: A tour guide for the world wide web. In *IJCAI (1)*, pp. 770–777, 1997.

[20] T. C. Kinnear and J. R. Taylor. *Marketing Research—an Applied Approach*. McGRAW-HILL, Inc., 4th edition, 1991.

[21] Y. Labrou and T. Finin. Yahoo! as an ontology: Using yahoo! categories to describe documents. In *Proc. CIKM*, pp. 180–187, Nov. 2–6 2000.

[22] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specifi cation. In *W3C Recommendation*, 1999.

[23] W. Litwin and A. Abdellatif. An overview of the multi-database manipulation language MDSL. *Proc. IEEE*, 75(5), pp. 621–631, May 1987.

[24] A. Luotonen. The common logfi le format. 1995.

[25] Microsoft. Microsoft Passport. Technical White Paper, Mar. 2001.

[26] C. Newell. AOL quietly launches magic carpet. In *eWeek.com*, 2002.

[27] B. Padmanabhan, Z. Zheng, and S. O. Kimbrough. Personalization from incomplete data: what you don't know can hurt. In *Knowledge Discovery and Data Mining*, pp. 154–163, 2001.

[28] M. Perkowitz and O. Etzioni. Adaptive Web sites. *CACM*, 43(8):152–158, Aug. 2000.

[29] G. Salton. *Automatic text processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, Reading, MA, USA, 1989.

[30] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, Sept. 1990.

[31] G. L. Somlo and A. E. Howe. Incremental clustering for profi le maintenance in information gathering web agents. In *Proc. AAMAS*, pp. 262–269, May 2001.

[32] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language. In *W3C Working Draft*, 2003.

[33] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. RFC 2413: Dublin Core metadata for resource discovery, 1998.

[34] H. Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proc. ACM SIGIR*, pp. 113–120. 2002.