Profile-Driven Data Management

Mitch Cherniack
Dept. of Computer Science
Brandeis University
Waltham, MA 02454
mfc@cs.brandeis.edu

Michael J. Franklin
Computer Science Division
University of California
Berkeley, CA 94720
franklin@cs.berkeley.edu

Stan Zdonik
Dept of Computer Science
Brown University
Providence, RI 02912
sbz@cs.brown.edu

Abstract

To support the popular vision of a semantic web, data management must be introduced as part of the web’s infrastructure. Traditional database systems follow the DBA-based model for data management — a DBA consults with clients about their intended uses of the database and sets data management policies (schema design, indexes, clustering etc.) accordingly. This model does not transfer well to the web where there are too many clients to consider in determining policy, and autonomous data sources prevent externally set data management policies from being imposed.

Profile-driven data management is a framework for determining and implementing data management policies upon the analysis of user data requirements specified in profiles. Profile-driven data management systems act as middleware intermediaries between users and the web, automatically offering data management services to thousands of clients by interpreting and reconciling their data needs, as specified in their profiles. In this paper, we introduce a language framework for expressing user profiles and outline an initial approach towards processing profiles expressed in this language in order to affect data management policies.

1 Introduction

Vast amounts of data populate the Internet, yet this data is largely unmanaged. Access to it involves a wide array of network resources including server capacity, bandwidth, and cache space. Applications compete for these resources with little overarching support for their intelligent allocation. To make matters worse, these resources are fairly autonomous and offer little or no outside control over how they are managed or allocated.

Application-blind resource use can lead to large inefficiencies, which in turn, limit scale. Consider a web-cache in which pages are kept with expiration times. A simple synchronization policy would eagerly refresh all pages as soon they expire. A more application-aware policy would refresh only those pages that are popular, or those for which up-to-date information is most crucial, thereby saving resources that would be wasted by performing updates that are likely to produce little or no value.

Over the years, database management systems have honed mechanisms that exploit application-level knowledge in data management decisions. Indexes are one key example. If, say, a large number of applications access flight information by departure-city, then it makes sense to construct an index on departure-city. Traditionally, such decisions have been made by a human database administrator (DBA), whose job is to understand and prioritize the competing requirements of
multiple applications. The DBA makes data management choices that aim to balance these requirements in a way that maximizes the overall good of the user community. Recent work in auto-administration [CN98] is focused on automating some of this functionality, but even such approaches depend upon having a single point of feedback and control. In such systems, application requirements can be specified coarsely, if at all, since any problems that arise can be detected and acted upon quickly.

In the Internet, there can be no DBA. First, the data sources don’t allow it, and second the scale and the dynamic nature of the Internet make strict control (as exercised by the DBA) an impossibility. Thus, we take the view that in order to allow for intelligent data management in the Internet environment, the DBA role must be replaced by a more distributed, dynamic, less direct approach. For such an approach to work, the specification of user/application-level data requirements must now be made formal. We call such a formal specification a profile. Profiles are obtained from a community of users and leveraged by intelligent middleware that effects data management decisions similar to those that would be made by a DBA in a more traditional approach. Profile-driven data management, then, replaces the centralized control of DBA approach in large, dynamic networks of autonomous data sources such as Internet.

In this vision, profile-driven data management is a way to manage resources such as communication bandwidth, disk bandwidth, sever capacity, and cache space. The kinds of data management techniques that we believe will have relevance in this environment include caching, data staging, indexing, clustering, declustering, replication, synchronization, and pre-computation. These techniques can be used to improve performance or to improve the quality of the response to a data request. Performance benefits because data might end up nearer to its point of use or in a form that is better suited to its intended user. Quality improves because the middleware layer can bring to bear substantial information resources for making intelligent data delivery and filtering decisions. For example, consider a data service that emphasizes data from the most popular among the sites that store some data, thereby finding information that appears to be valued by many users.

This paper is a first step in this direction. We describe the notion of profile-driven data management in Section 2. Section 3 introduces the language framework for profiles complete with a formal semantics. Section 4 discusses some issues and preliminary solutions to the problems of processing profiles to achieve our data management goals. We present related work in Section 5 and conclude in Section 6.

2 Profile-Driven Data Management

As introduced in the previous section, profile-driven data management is based on the premise that high-level specifications of user interest (i.e., profiles) can be used in a constructive way to improve the performance and effectiveness of Internet data sources.

2.1 An Architecture

The architecture shown in Figure 1 is our view of a framework in which to deploy profile-driven data management. In this architecture, the middle layer is where most of the data management functionality resides. This layer receives profiles from a community of users and processes those profiles with respect to a set of data sources (possibly the Web). The profiles are used to make data management decisions. While they resemble queries in many respects, they do not necessarily
Figure 1: An Architecture for Profile-Driven Data Management

return any data. They simply identify data of interest – data that might be requested in some future query.

The figure identifies three distinct phases in our architecture. The first is data gathering, the process of acquiring some portion of the data of interest from the data sources. This process involves finding appropriate data sources, extracting data, and returning it to the middle layer in the proper format. The second phase is the data management task of the middle layer. This could include, for example, caching and indexing. The third phase is the delivery of data to the user community. Such delivery can be in response to queries or it could be a push-based multicast from which users filter answers of interest.

2.2 Example Applications

In this paper, we discuss two applications of profile-driven data management: data recharging and data freshening. While these applications are quite different, they both share a common need to allocate some limited resource to a set of external data objects.

Data recharging is best understood in the context of a mobile computing environment, although the basic techniques are also relevant in the wired case. Mobile computers have two fundamental renewable resources, power and data. Recharging power from the power grid is a simple matter for any user. The goal of data recharging is simply to make recharging data from the data grid equally simple. Based on a user profile, the data recharging middleware will gather data of interest to a user, and when the user next connects to the network, this data will be delivered to the mobile machine. In this case, the profile would likely depend on some aspect of the user’s context (e.g., location, workflow). The data delivered would depend on available resources on the mobile machine (e.g., memory, applications). The order in which items are delivered might also matter if unplanned disconnection was possible.

In the mobile environment, computers are often memory-limited. Thus, given a large amount of potentially interesting data, a data recharging system must make decisions about what subsets of this data are to be allocated to the limited memory resource.

Data freshening [CGM00a] is done relative to a collection of cached objects or replicas. If the primary copy is updated, then the cached copy must be refreshed. This could be done on access (as in most proxy caches) or it could be done proactively based on expected usage and expiration times. In data freshening, the latter approach is chosen. If there are many cached objects and a refresh operation requires significant resources, then we would want to refresh only a subset of the cached
PROFILE Investor

DOMAIN
NA = http://www.nasdaq.com
NY = http://www.nyse.com
CNN = http://www.cnn.com

UTILITY
U (NA) = IF (AGE < 30 min) THEN 5 ELSE 0;
U (NY) = IF (AGE < 20 min) THEN 3 ELSE 0;
U (CNN) = IF (AGE < 10 min) THEN 1 ELSE 0

END

(a): A URL-Based Profile

PROFILE Traveler

DOMAIN
R = related:www.hertz.com
S = Logan-Airport AND downtown AND "shuttle schedule"
D = "Downtown Boston" AND hotel AND "directions from Logan Airport"

UTILITY
U (R) = UPTO (2, IF (#D > 0) THEN 1 ELSE 0, 0);
U (S) = UPTO (1, 2, 0);
U (D) = UPTO (1, 1, 0)

END

(b): A Search-Engine-Based Profile

Figure 2: Two Example Profiles

objects. Choosing this subset is called the refresh problem and can be done more intelligently if we have a set of user profiles. This data staging problem is another candidate for profile-driven data management.

If freshening a data item consumes some finite amount of network resource (bandwidth, server capacity), it might not be practical to refresh everything during each refresh cycle. Thus, choosing what items should be refreshed at a given point in time reduces to a resource allocation problem.

2.3 Profiles

The fundamental enabler to this approach is the profile. Profile languages must be expressive enough to make important distinctions, yet must be simple enough to make reasonable processing algorithms possible. The description of a framework for specifying profile languages is the main contribution of this paper.

To illustrate how profiles can influence data management, consider the profiles of Figures 2a and 2b. First, notice that the profile specifications are broken into two parts. The Domain clause defines and names sets of objects of interest (domain sets). The Utility clause specifies the relative values of these objects of interest. The Utility clause distinguishes our of notion of profiles from those typically found in publish/subscribe systems. We will explain the intuitive meaning of these profiles here, and present profile languages more formally in Section 3.

The Investor profile of Figure 2a defines three domain sets, one for each of three interesting sources of stock market information: the New York Stock Exchange (NY), NASDAQ (NA) and CNN
(CNN). In this case, the domain sets each consist of a single web page: the home page for the given stock market source. The Utility clause of this profile tells us that the value of the NYSE web page in our cache is 3 provided that it is less than 20 minutes old (i.e., it has been freshened in the past 20 minutes) and 0 if it is any older. This is motivated by the fact that NYSE releases new stock data to the public every 20 minutes. A similar utility value is associated with the NASDAQ web page. The CNN web page must include recent news to have value (no more than 10 minutes old), and even then, its value to the investor is less than the those of web pages for NYSE and NASDAQ.

Figure 2b shows a simple profile that might drive data recharging for a PDA owned by a traveler who is about to travel to Boston. The traveler needs to get from Logan Airport to her hotel in downtown Boston, and can make the trip either by rental car, or by taking a shuttle. To take a shuttle, the traveler requires a shuttle schedule for any company offering a shuttle service. To take a rental car, the traveler needs to know the rates, policies, etc. for a rental car company (or more than one company, so that she can compare them), and also needs directions from the airport to downtown. Even if the traveler takes a shuttle and not a rental car to get downtown, directions still serve some use as they tell her a little bit about how to get about the city.

Profile Traveler specifies the data needs of the traveler just described. The Domain clause of this profile identifies 3 domain sets. R is a set of rental car company web pages that (presumably) offer details about rates and policies. S is a set of shuttle schedules for shuttles heading to Downtown Boston from Logan. And objects in D are web pages, text files etc. that give directions from Logan to hotels in Downtown Boston. Note that unlike the investor profile that used URL’s to specify domain sets, the traveler profile specifies the sets associatively with expressions that resemble inputs to a search engine. (In fact, the domain set expressions in this profile would be acceptable inputs to Google [Goo].) As with a query, associative specifications of domain sets free users from finding the data that interests them (the traveler may not care whether Logan-Downtown directions are generated by Maps-On-Us [Map] or found in a hotel web page), and makes the profile processor responsible for finding data of interest.

The Utility clause of the traveler profile specifies the data values and dependencies described earlier. Consider the value of shuttle schedules (S). The expression, \( \text{Upto} (1, 2, 0) \), says that a shuttle schedule that is included in the set of objects chosen to charge a device has value 2, but once a schedule is in the charge, every other shuttle schedule in the charge has no value. Thus, \( \text{Upto} (x, y, z) \) is a threshold operator; indicating that upto \( x \) objects have a value of \( y \) each, and every object past the threshold \( x \) have a value of \( z \) each. Direction objects (i.e., objects from domain set, D) are similarly valued; the first in a charge has a value of 1 and every one thereafter has a value of 0. These expressions reflect the traveler’s needs. The traveler has need for at most, one shuttle schedule, and at most, one direction object. But a single shuttle schedule has worth exceeding that for the direction object because by itself, it has enough information to get the traveler to his hotel.

The utility of rental car objects (R) requires more complex expression. As with shuttle schedules and direction objects, R objects have value up to a threshold; only the first 2 to be stored in a PDA cache have any value. The value of these first \( R \) objects is expressed with the conditional expression,

\[
\text{IF} \ (\#D > 0) \ \text{THEN} \ 1 \ \text{ELSE} \ 0.
\]

This expression says that if the number of \( D \) objects \( (\#D) \) that have been included in the charge is at least 1, then the rental car object has value 1. This reflects the traveler’s data needs which say that the traveler can only take a rental car to Downtown Boston if she has directions, thereby expressing a utility value dependency between \( R \) and \( D \) objects. That the threshold for rental car
objects is 2 indicates that having more than one web page permits a comparison of rates, but having more than 2 is not worth the memory it occupies in the cache.

The profiles of Figure 2 are overly simplified, but are intended to show that user data needs can be formally expressed. In practice, we expect profiles to describe many more domain sets with more complex utility value expressions. We consider it unlikely that most users will write profiles manually (the same could be said for SQL). Instead, we expect that a profile-generation system with good interfaces could support users to this end. Such a system could rely on libraries of parameterized profiles that are built and extended by experts. The key idea is a that declarative profile language is needed to facilitate this process. Finally, we recognize that profile-driven data management will have to reconcile the data needs of thousands of users, as specified with thousands of profiles. We consider techniques for processing multiple profiles for the purposes of data management in Section 4.

3 A Profile Language Framework

A user’s profile describes what data objects are considered interesting and how useful these data objects are relative to each other. In this section, we argue that the nature of the data being profiled influences how the Domain and Utility clauses of a profile can and should be specified. We then describe a profile language framework that enables the specification and generation of a family of profile languages that vary according to the target data they are intended to profile. We present the framework using a parameterized grammar [BB95]: a grammar formalism that permits the designation of select nonterminals as parameters.

3.1 Profile Language Syntax

Figure 3 shows a parameterized grammar that specifies a framework for the syntax of profile languages. A parameterized grammar (sometimes called a modular grammar [DC90]) is a partially specified grammar that includes certain nonterminals (the parameters) that are never reduced. That is, a grammar’s parameters appear in the bodies but never in the heads of production rules. Such grammars are partial language specifications; instantiation of parameterized grammars with the missing production rules “completes” the language definition. Parameterized grammars are supported in many modern compiler-compiler tools (e.g., PRECC [BB95]) and enable the generation of modular (i.e., composable) parsers. Given that different parameter instantiations produce different language specifications, the parameterized grammar is suitable for specifying a language framework.

Our profile language framework is motivated by two observations:

1. The nature of the data being profiled affects how interesting data can be identified in a profile’s domain clause.

   Given the similarity between the domain clause of a profile and a query, this is not surprising. Every data model is queried by a different query language, as there is no singular query language that universally specifies all data.

2. For the most part, utility clauses can be uniformly specified in spite of the variations in how domains get specified.
Profile  ::=  PROFILE IDENT DomainClause UtilityClause END

DomainClause  ::=  DOMAIN DomainSetEqn ' ; ' ... ' ; ' DomainSetEqn
UtilityClause  ::=  UTILITY UtEqn ' ; ' ... ' ; ' UtEqn

DomainSetEqn  ::=  IDENT '=' (DomainSetExp)

UtEqn  ::=  U '(' IDENT ')' '=' UtExp

UtExp  ::=  INTEGER
| IF Conds THEN UtExp ELSE UtExp
| UPTO '(' INTEGER ',' UtExp ',' UtExp ')

Conds  ::=  SCond
| Conds AND SCond
| Conds OR SCond
| NOT '(' Conds ')

SCond  ::=  ⟨ObjectExp⟩ RelOp INTEGER
| ⟨ObjectExp⟩ RelOp ⟨ObjectExp⟩
| DomExp RelOp INTEGER
| DomExp RelOp DomExp

DomExp  ::=  '#' IDENT
| '#' IDENT '[' SCond ']
| '#' IDENT '[' JCond ']

JCond  ::=  '(' IDENT '.' ⟨ObjectExp⟩ ')' RelOp ⟨ObjectExp⟩

RelOp  ::=  '=' | '!=' | '<' | '>' | '<=' | '>='

Figure 3: A Parameterized Grammar for Profile Languages

The variation in domain specifications means that there is not one, but many profile languages. But
the near-uniformity of utility specifications means that a common language framework is possible,
as are processing strategies that can be applied to profiles expressed in any language resulting from
this framework. We discuss one such processing strategy in Section 4.

The parameterized grammar of Figure 3 resembles a standard BNF grammar, where UPPERCASE strings denote tokens, and Mixed-case strings denote nonterminals. Profiles are denoted in
this grammar with the nonterminal, Profile. This grammar also uses two parameter nonterminals
(denoted with ⟨angle brackets⟩): ⟨DomainSetExp⟩ and ⟨ObjectExp⟩.
3.1.1 Domain Clauses

The Domain clause (DomainClause) of profile languages consists of equations of the form,

\[ D = \langle \text{DomainSetExp} \rangle \]

such that \( D \) names a domain set: a set of objects whose utility values are determined uniformly, and \( \langle \text{DomainSetExp} \rangle \) is an expression defining the contents of this set. Domain set specifications serve two purposes:

- Domain set names form a vocabulary for use in specifying utility values.
- Domain set expressions drive data gathering by identifying objects of interest.

The nonterminal, \( \langle \text{DomainSetExp} \rangle \) is a grammar parameter. There are no production rules defining the language of domain set expressions because the nature of the data being profiled determines what this language should be. Consider the example profiles of Figure 2a and 2b. Domain set expressions in the former are expressed using URL’s, which can be used to identify any data item accessible via the HTTP protocol (html pages, pdf files, jpeg images etc.). The variety of data type expressible by URL is offset by the navigational flavor of the specification — the author of the profile must know the locations of the data that interests him and must edit his profiles should these locations change. On the other hand, domain set expressions in the Traveler profile are specified with a language resembling the input language for a search engine.\(^1\) The downside of this approach is that search engines can return spurious results, and are limited to finding data adhering to certain formats (e.g., Google indexes html and pdf files, but not ps files). But, the declarative flavor of these expressions means that the user does not have to know exact locations of the data that interests him, nor does he have to edit his profile if the data sources that contain interesting data should change.

The above examples show that the nature of the data being profiled affects how domain sets can be specified; profiles must specify URL’s if they want access to any kind of data available off of the web, but can use more associative search-engine inputs if they are willing to confine their interest to data of a few different formats. The point is illustrated further when one considers structured data such as relations or XML. If a user is willing to confine her attention to structured data, she can specify a domain set with a query (e.g., SQL for relational data, or XML-QL [FDL+99] for XML data).\(^2\)

Figure 4 shows an example profile that expresses domain sets over XML data with XML-QL. (In the interest of space, we only show the XML-QL query for the first domain set.) Domain set \( P \) identifies a set of XML elements naming schools that have database faculty openings. Note that unlike the profiles of Figures 2a and 2b that define domain sets as sets of externally located web pages, the objects in this domain set are constructed as a result of extracting from external data sources. The ability to synthesize new data objects by extracting from or integrating data found in external sources rather than accepting data objects “as is” shows another advantage of using a query language to specify domain sets.

\(^1\)In fact, these domain set expressions would be acceptable inputs to Google [Goo].

\(^2\)Actually, query languages as they stand now are ill-suited as languages for expressing domain sets, because they demand explicit naming of external data sources (i.e., in the FROM clause of SQL queries). Others have proposed ways to make the FROM clause more declarative (e.g., in the context of NiagaraCQ [CDTW00a] for XML-based queries). While an important issue, we do not consider it further here.
PROFILE Academic

DOMAIN
P = WHERE
  <Faculty-Announcement>
  <School>$S< / >
  <SoughtRank>$R< / >
  <Area>Databases< / >
  < / > IN "www.cs.wisc.edu/dbworld.xml"

CONSTRUCT
  <Position>
  <School>$S< / >
  <$R< / >
  < / >

V = {VLDB authors and their papers written by faculty at schools listed in I}

UTILITY
U (P) = IF ((Position.Rank = 'Full') OR (Position.Rank = 'Assoc')) THEN 5 ELSE 3;
U (V) = IF (#P [P.School = School] > 0) THEN 3 ELSE 0

END

Figure 4: Academic: An XML-QL-Based Profile

To achieve flexibility in how to specify domain sets, our framework uses a parameter nonterminal, \(<\text{DomainSetExp}\)\), to represent domain set expressions. A complete profile language can be specified by instantiating this parameter with a subgrammar that specifies the structure of this nonterminal (e.g., with the grammar for XML-QL), and by instantiating the parameter, \(<\text{ObjectExp}\)\), which we discuss below.

3.1.2 Utility Clauses

The Utility clause (UtilityClause) of profile languages consists of equations of the form,

\[ U (D) = U\text{tExp}. \]

Each utility equation establishes the utility value of objects of a particular domain set (i.e., \(U (D) = 3\) says that each object of domain set \(D\) has value 3). Note that it is possible, depending on what language is used to express domain sets, for a given data object to belong to more than one domain set. For example, a web page that offers information about rental car rates and also gives directions from Logan Airport to downtown Boston would belong to domains \(R\) and \(D\) of profile Traveler in Figure 2b. Such objects have a utility value that is the sum of the values specified for each of their associated domain sets, following the intuition that objects that meet multiple needs have added value.

As we saw with the profiles of Figure 2, utility expressions need not be integers, but can be the integer results of evaluating more complex expressions that consider what other objects have been allocated the resource being managed. The Traveler profile of Figure 2b used the \(\text{UPTO}\) operator to express utility values with \(\text{thresholds}\). \(\text{UPTO}\) expressions can be nested to an arbitrary depth, enabling the expression of stepwise decaying (or growing) utility values. For example,

\[ U (D) = \text{UPTO} (k_1, v_1, \text{UPTO} (k_2, v_2, v_3)) \]
say that the first \( k_1 \) objects of domain set \( D \) to be allocated a given resource are worth \( v_1 \) each, the next \( k_2 \) objects of \( D \) to be allocated the resource are worth \( v_2 \) each and all objects thereafter are worth \( v_3 \).

UPTO allows the value of an object to be conditional on what other objects from the same domain have been allocated a given resource. Sometimes, object utility values must be conditioned on the number of objects from other domains that have been allocated the given resource, or on other conditions that are specific to the object whose value is being determined (such as its age). So as to express such utility values, we permit utility values to be expressed conditionally using IF ... THEN ... ELSE:

\[
U(D) = \text{IF Cond THEN UtExp ELSE UtExp.}
\]

Conditions (Cond) can be complex (e.g., conjunctions) or simple (SCond), and simple conditions can in turn be object or domain conditions. An object condition is a condition on the object whose value is being determined, such as the “AGE < 20 MIN” condition used in profile Investor from Figure 2a. Profile Academic of Figure 4 also uses object conditions in specifying the value of faculty announcement objects (P). The value of a faculty announcement (P) object is 5 if the position is for senior faculty and 3 otherwise.

A domain condition expresses a domain dependency: a condition based on the number of objects of some domain that have been allocated the resource being managed using the profile. We saw such a dependency with the utility equation for domain R in the Traveler profile of Figure 2b, which included the expression, \#D > 0. This expression was used to indicate that the value of R objects depends on the presence of D objects included in the data charge. Profile Academic of Figure 4 uses a conditional expression in its utility equation for V objects: XML elements that list the titles of VLDB papers written by authors at schools referred to in P, as well as the paper authors and their affiliations. The condition,

\[
\# P [P.School = School] > 1
\]

is true of a given VLDB paper (v, a V object) if there is at least one faculty announcement from the school that has an author of v on its faculty that has been allocated the resource. (We refer to the condition, “P.School = School” as a join condition because it compares values extracted from two different objects: a P object and a V object. Taken in its entirety, the utility equation for V attaches high value to papers that have been written by faculty members that are affiliated with schools whose announcements have been allocated the resource.

No matter the choice of language to specify domain sets, utility equations are largely uniform. The only variation is in the expression of object expressions that are compared in object conditions and join conditions. Such expressions are largely dependent on the choice of language for specifying domain sets. For example, if a domain set is specified by a URL, then the only possible object expressions are those involving metadata that are returned with an object when that object is retrieved: age, expiry date/time, etc. For domain sets of structure objects specified with queries, an object expression can be the result of extracting on a field of the object (e.g., School). Because of the nature of object expressions depends on the data being profiled, we have made the nonterminal representing object expressions, \( \langle ObjectExp \rangle \), a second parameter to the grammar.

### 3.2 Formal Semantics

Let \( L(S, E) \) be the profile language that results from instantiating our profile language framework with some language of domain set expressions, \( S \), and some language of object expressions, \( E \), and
let $\Omega$ be the universe of data objects that this profile language is able to profile (e.g., $\Omega$ might consist of all XML files retrievable off of the Internet if $S$ were defined as an XML-based query language.) We assume that $E$ and $S$ each have a formal semantics as defined below.

**Assumed semantics of parameters:** We assume that for any expression $e$ that is well-formed according to $E$, its semantics is a function,

$$\llbracket e \rrbracket: \Omega \rightarrow \Omega$$

that maps objects in $\Omega$ to other objects in $\Omega$. Similarly, we assume that for any domain set expression, $s$ that is well-formed according to $S$, its semantics is a predicate,

$$\llbracket s \rrbracket: \Omega \rightarrow \mathit{Bool}$$

that returns true of any object in $\Omega$ that belongs to the set denoted by $s$.

If a domain $D$ has been specified by the data set expression, $s$, we write $\sigma_D (O)$ to denote the set of all $D$-objects

$$\sigma_D (O) = \{ x \in O \mid \llbracket s \rrbracket (x) \}$$

for any $O \subseteq \Omega$.

**Semantics of profiles:** Let $P$ be any profile expressible in $L(S, E)$, and let $D = \{ D_1, \ldots, D_n \}$ be the set of domains it defines such that for any $D_i \in D$, the utility value of objects in $D_i$ is specified by the utility equation,

$$U(D_i) = UT_i.$$  

Then the semantics of profile, $P$ is a function,

$$\llbracket P \rrbracket: 2^\Omega \rightarrow \mathit{Int},$$

that maps any set of objects to the value of that set, as specified by the utility equations in $P$. More formally, for any set, $O \subseteq \Omega$,

$$\llbracket P \rrbracket (O) = \sum_{1 \leq i \leq n} (\llbracket UT_i \rrbracket (\sigma_{D_i}(O), O)),$$

such that the semantics of a utility expression, $UT_i$, is a function on pairs of sets of objects, $(S_1, S_2)$ as defined by the following equations:

$$\llbracket i \rrbracket (S_1, S_2) = |S_1| \cdot i \text{, for integer } i$$

$$\llbracket \text{UPTO} (k, UT, UT') \rrbracket (S_1, S_2) = \begin{cases} \llbracket UT \rrbracket (S_1, S_2), & \text{if } |S_1| \leq k \\ \llbracket UT \rrbracket (S_1', S_2) + \llbracket UT' \rrbracket (S_1'', S_1) & \text{for any } S_1', S_1'' \subseteq S_1 \text{ s.t. } |S_1'| = k, \ S_2 = S_1 - S_1'', \text{ otherwise} \end{cases}$$

$$\llbracket \text{IF } C \text{ THEN } UT \text{ ELSE } UT' \rrbracket (S_1, S_2) = \begin{pmatrix} \llbracket UT \rrbracket (\{ x \in S_1 \mid \llbracket C \rrbracket (x, S_2) \}, S_2) \\ \llbracket UT' \rrbracket (\{ x \in S_1 \mid \neg (\llbracket C \rrbracket (x, S_2)) \}, S_2) \end{pmatrix}$$
Table 1: Semantics of Conditional Expressions

Semantics of conditions: The semantics of utility expressions depends upon the semantics of conditions (Conds), which in turn depends on the semantics of simple conditions (SCond), domain expressions (DomExp), join conditions (JCond), and the instantiating language of object expressions ((ObjectExp)). The semantics of these expressions is summarized in Table 1.

Using the Semantics Equations to Determine the Value of an Object Set: The semantics equations described above can be used to determine the value of an object set relative to a profile. To illustrate, consider the Traveler profile of Figure 2b defined in profile language, $L(S, E)$, such that the semantics of $S$ establishes that for some set of objects,

$$O = \{r_1, r_2, r_3, d_1\}$$

that $\sigma_R \{r_1, r_2, r_3\}$, $\sigma_D \{d_1\}$, and $\sigma_S \emptyset$. Then the value of $O$ with respect to Traveler is:

$$\langle Traveler \rangle (\{r_1, r_2, r_3, d_1\}) = (\langle UT_R \rangle (\{r_1, r_2, r_3\}, O)) + (\langle UT_S \rangle (\emptyset, O)) + (\langle UT_D \rangle (\{d_1\}, O))$$

$$= 2 + 0 + 1 = 3$$

as the subderivations shown below show.
\[ \llbracket UT_h \rrbracket \{r_1, r_2, r_3\}, O \]
\[ = \llbracket\text{Upto} (2, \text{IF } (#D > 0) \text{ THEN } 1 \text{ ELSE } 0) \rrbracket \{r_1, r_2, r_3\}, O \]
\[ = (\llbracket\text{IF } (#D > 0) \text{ THEN } 1 \text{ ELSE } 0 \rrbracket \{r_1, r_2\}, O) + (\llbracket 0 \rrbracket \{r_3\}, O) \]

Note that \( \llbracket #D \rrbracket (x, O) = |\sigma_D (O)| = 1 \) for any object, \( x \). Thus, this reduces to:
\[ = (\llbracket 1 \rrbracket \{r_1, r_2\}, 0) + (\llbracket 0 \rrbracket (0, O))) + (\llbracket 0 \rrbracket \{r_3\}, O)) \]
\[ = (1 \cdot 2) + (0 \cdot 0) + (0 \cdot 1) = 2 \]

\[ \llbracket UT_5 \rrbracket (\emptyset, O) \]
\[ = \llbracket\text{Upto} (1, 2, 0) \rrbracket (\emptyset, O) \]
\[ = \llbracket 2 \rrbracket (\emptyset, O) \]
\[ = 2 \cdot 0 = 0 \]

\[ \llbracket UT_5 \rrbracket \{d_1\}, O \]
\[ = \llbracket\text{Upto} (1, 1, 0) \rrbracket \{d_1\}, O \]
\[ = \llbracket 1 \rrbracket \{d_1\}, O \]
\[ = 1 \cdot 1 = 1 \]

4 Profile Processing

What is common to the data recharging and data freshening applications described in Section 2 is the periodic need to choose some subset of known objects to be allocated a limited resource. In the case of data recharging, the limited resource is the memory cache in the device getting charged with data. In the case of data freshening, the limited resource is the available bandwidth (measured in bytes per fixed time unit) required to retrieve fresh copies of objects. In both cases, profiles can assist with the task of choosing a subset of objects to be allocated the resource and that have high value to the user(s).

In this section, we consider various ways that profiles can be processed to decide on an appropriate allocation of some resource. We begin in Section 4.1 by defining the resource allocation problem (RAP) formally as a constraint problem. In Section 4.2, we present a greedy heuristic to address it, and in Section 4.3, we discuss current work studying knapsack-based solutions. Then, in Section 4.4, we consider how to cope with multiple profiles that are contending for the same limited resource, and present a profile combination algorithm that combines multiple profiles into a single “superprofile” that can then be used as input to the greedy algorithm for RAP.

4.1 RAP: The Resource Allocation Problem

Formally, RAP can be stated as follows:

**Definition 4.1 (Resource Allocation Problem)** Given:

1. a finite set of known objects, \( O \), such that for any object, \( o \in O \), \( s(o) \) is its size (in bytes),

2. a profile, \( P \) that specifies for any subset, \( O' \subseteq O \), its value, \( v(O') \) (note: \( v(O') = \llbracket P \rrbracket (O') \)),

While admittedly a simplified notion of capacity, we work under the assumption that other factors that influence the amount of data that can be updated in a fixed time period such as network contention, server loads etc. are canceled out when comparing the expected bandwidth required to update different objects.
ALGORITHM GREEDY (P: Profile, O: {Object}, A: {Object}, C: INT): Object
BEGIN
    max_added_value := 0
    res := NULL
    For every o in O DO
        IF (SIZE (o) < C) AND (Value (o, A) / SIZE (o)) > max_added_value THEN
            max_added_value := Value (o, A) / SIZE (o)
            res := o
        RETURN o
    END

Figure 5: Pseudocode for the Greedy Algorithm for RAP

3. a device capacity, C (in bytes)

Determine the subset \( O' \subseteq O \) that satisfies the constraint,
\[
\sum_{o \in O'} (s(o) \leq C)
\]

and that maximizes the value function, v.

This problem resembles the knapsack problem, but for the nature of the value function which maps sets rather than individual objects to their values. The problem must be formulated in this way because the value of an individual object can vary according to its context (i.e., the other objects that have been allocated the resource) due to domain dependencies and thresholds. We present a simple greedy heuristic algorithm for RAP, with the purpose of showing that profiles can be processed in a meaningful way to allocate limited resources. We then discuss ongoing work that considers more sophisticated strategies (e.g., based on algorithms targeting variations on the knapsack problem such as precedence constrained knapsacks).

4.2 A Greedy Algorithm

The greedy algorithm for RAP works as follows. In choosing a subset of objects to be allocated a resource, we repeatedly cycle through the entire set of objects looking for the object:

- that has maximum added value (as measured in utility value per byte) given the set of objects that have been allocated to the resource thus far, and

- whose size does not exceed the remaining available capacity of the resource.

The pseudocode for this algorithm is given in Figure 5. The arguments to this algorithm include a profile, P, a set of objects that can be allocated the resource, O, a set of objects that have already been allocated the resource, A, and a resource capacity C (measured in bytes). In the case of data recharging, GREEDY could be invoked numerous times to return a set of objects to deliver to a device during its charge. Each call of GREEDY would return one of these objects, o, and before invoking GREEDY again, o would be removed from O and added to A, and the available capacity of the device (C) would be decremented by o's size.
GREEDY assumes the existence of a function, Value, that determines the worth of an object \( o \) relative to an existing set of objects, \( A \). More precisely, this is the difference between the value of \( A \) with \( o \), and the value of \( A \) alone. I.e., 

\[
\text{Value}(o, A) = (\|P\| (\{o\} \cup A)) - (\|P\| (A)),
\]

though for performance reasons, Value is not likely to be computed in this way.

To demonstrate the execution of this algorithm, consider its use in a data recharging scenario with profile Traveler of Figure 2b. Suppose that before recharging the traveler’s PDA, data gathering has returned a set of objects, \( O \) from which to choose such that:

\[
O = \sigma_R(O) \cup \sigma_S(O) \cup \sigma_D(O) \quad \text{and}
\]

\[
\sigma_R(O) = \{r_1, r_2, r_3\},
\quad \sigma_S(O) = \{s_1, s_2\},
\quad \sigma_D(O) = \{d_1, d_2\},
\]

such that every object has size 10Kb, and the device cache has room for 4 objects, or 40Kb. The initial call made to GREEDY then, is:

\[
\text{GREEDY (Traveler, O, \emptyset, 40Kb)}
\]

and a trace of the resulting execution is summarized by the table below.

Each row in the table below represents a call to GREEDY, and consecutive rows denote consecutive calls. The table has 5 columns. The first column indicates which call to GREEDY is described. The second column shows what objects have been allocated (\( A \)) at the time of the call. Note that in the context of data recharging, \( A \) shows the objects that have been chosen thus far to be delivered as part of the charge. The third column, comprising 3 values, lists all available objects by domain set, where an available object is one that is eligible to be added to the current allocation. For each domain set, the set of available objects belonging to that set are accompanied by a value, in parentheses, that denotes what value objects from that domain set would add to the allocation if selected. The fourth column shows which object is added as a result of the call to GREEDY\(^4\), and the fifth column shows the value of \( A \) as a result of adding this object.

<table>
<thead>
<tr>
<th>Call #</th>
<th>A: Allocated Objects</th>
<th>( O ): Available Objects (Added Value)</th>
<th>Object Chosen</th>
<th>Result Utility Value of A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>( r_1, r_2, r_3(+0) ) ( s_1, s_2(+2) ) ( d_1, d_2(+1) )</td>
<td>( s_1 )</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>( {s_1} )</td>
<td>( r_1, r_2, r_3(+0) ) ( s_2(+0) ) ( d_1, d_2(+1) )</td>
<td>( d_1 )</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>( {s_1, d_1} )</td>
<td>( r_1, r_2, r_3(+1) ) ( s_2(+0) ) ( d_2(+0) )</td>
<td>( r_1 )</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>( {s_1, d_1, r_1} )</td>
<td>( r_2, r_3(+1) ) ( s_2(+0) ) ( d_2(+0) )</td>
<td>( r_2 )</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>( {s_1, d_1, r_1, r_2} )</td>
<td>( r_3(+0) ) ( s_2(+0) ) ( d_2(+0) )</td>
<td>( - )</td>
<td>( - )</td>
</tr>
</tbody>
</table>

One advantage of the greedy algorithm is that it is easily modified to cope with disconnection when applied to data recharging. Specifically, to make data recharging as robust as battery recharging,\(^4\)

\(^4\)Note that because object sizes are uniform in this simple example, each call of GREEDY is simply choosing a domain set from which the next object should be drawn from. The actual object chosen from this domain set is selected arbitrarily.
it is necessary to design allocation strategies that ensure that a device's memory cache has been charged with objects appropriately even when a premature disconnection from the network does not permit the cache to be filled. Because each time it is called, the greedy algorithm chooses the most valuable object relative to the present allocation of objects, the order in which it chooses objects is also an appropriate order for the delivery of objects, ensuring a valuable cache, even if the algorithm is interrupted prior to filling the cache.

On the other hand, the greedy algorithm is sub-optimal in certain cases. Suppose that GREEDY is used in a data freshening context with the Investor profile of Figure 2a to determine which web pages in a cache to update, given that one page can be updated every 10 minutes. In this case, we'll assume that the cache, O, consists of the CNN web page (cnni), the NYSE web page (nyj) and the NASDAQ web page (nak) such that i, j and k reflect the ages of these pages in the cache in minutes (i.e., the elapsed time since the last refresh of the given web page). Again, for simplicity, we assume each web page has size 10Kb. As one object can be updated at a time, the call made to determine the next object to update is: GREEDY (Investor, O, O, 10Kb).

A table tracing the result of calling GREEDY 4 times, once every 10 minutes starting at hour x, is shown below. For this example, we assume GREEDY is first called with an existing cache that has a 20 minute old NASDAQ web page (nak0), a 10 minute old NYSE web page (ny0), and a 30 minute old CNN web page (cnn0). Thus, when GREEDY is first called, the value of the cache with respect to the Investor profile is 8, reflecting the fact that the NASDAQ and NYSE pages are young enough to have non-zero utility values (5 and 3 respectively), but that the CNN page is stale and therefore worth 0.

<table>
<thead>
<tr>
<th>Call Time</th>
<th>A: Allocated Objects</th>
<th>O: Available Objects (Added Value)</th>
<th>Object Chosen</th>
<th>Result Utility Value of A</th>
</tr>
</thead>
<tbody>
<tr>
<td>x : 00</td>
<td>{nak0, ny0, cnn0}</td>
<td>na0 (+0) ny0 (+0) cnn0 (+1)</td>
<td>cnn0</td>
<td>9</td>
</tr>
<tr>
<td>x : 10</td>
<td>{nak0, ny0, cnn10}</td>
<td>na0 (+5) ny0 (+3) cnn0 (+1)</td>
<td>na0</td>
<td>5</td>
</tr>
<tr>
<td>x : 20</td>
<td>{nak10, ny30, cnn20}</td>
<td>na0 (+0) ny0 (+3) cnn0 (+1)</td>
<td>ny0</td>
<td>8</td>
</tr>
<tr>
<td>x : 30</td>
<td>As at time x : 00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At the first call at time x : 00, the greedy algorithm examines which update would add the most value to the page. As an update to cnn is the only update that adds any value (given that cnn is the only stale page), it is selected and freshened. At time x : 10, the algorithm is invoked again and now every page in the cache is 10 minutes older. As a result, all pages are now stale and the most valuable page to freshen is na. When the greedy algorithm is invoked at time x : 20, cnn and ny are the only stale pages and the latter is chosen to be freshened. At time x : 30, we have the same situation we had at time x : 00 and the cycle repeats itself. Thus, the average value of the cache is

\[
\frac{9 + 5 + 8}{3} = 7.33.
\]

The greedy algorithm is sub-optimal in this case. To illustrate, we show a freshening policy that results in a higher average cache value than does greedy. This policy ignores the cnn web page, and alternately freshens na (beginning at time x : 00) and ny (beginning at time x : 10). The table below illustrates the contents of the cache given this freshening policy.
<table>
<thead>
<tr>
<th>Call Time</th>
<th>A: Allocated Objects</th>
<th>Object Chosen</th>
<th>Result Utility Value of A</th>
</tr>
</thead>
<tbody>
<tr>
<td>x : 00</td>
<td>{na_20, ny_{10}, cnn_{30}}</td>
<td>na_0</td>
<td>8</td>
</tr>
<tr>
<td>x : 10</td>
<td>{na_{10}, ny_{20}, cnn_{40}}</td>
<td>ny_0</td>
<td>8</td>
</tr>
<tr>
<td>x : 20</td>
<td>As at time x:00 except that cnn continues to age</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average value of this cache is 8.

### 4.3 Other Heuristics

As this example shows, **GREEDY** is a heuristic-based approach that is likely to be sub-optimal in many situations. In ongoing work, we are exploring the use of algorithms for solving the Knapsack Problem for profile evaluation. In particular, a restricted version of our profile language can be mapped into a *precedence-constrained* knapsack problem (PCKP), in which the values of the items placed into the knapsack can have dependencies on the other items that have been placed into it. Such precedence constraints can be used, for example, to model situations where some item B has value only if some other item A is also chosen. Unfortunately, this approach does not support the full generality of our profile model. For example, it cannot capture the case where items A and B are mutually dependent.

Mapping the profile evaluation problem to PCKP allows us to employ a dynamic programming algorithm to obtain an optimal solution. Even this solution, however, can be quite expensive in terms of running time \(O(n \times V^2)\), where \(n\) is the number of items, and \(V\) is the highest possible aggregate value for those items). One way to reduce the cost of this algorithm, is to approximate it, by chopping off lower order bits of the aggregate values, thereby reducing the number of possible values [Pap94]. Early results using this approach [Tom00] indicate that the approximate PCKP algorithm can produce much better results than the **GREEDY** algorithm, even with fairly coarse approximation factors (i.e., many bits removed from the value calculation). In addition, the **GREEDY** algorithm can be applied to the set of items that results from approximate PCKP, so that the items can be delivered by decreasing value to cope with potential disconnection for the case of Data Recharging. This work, however, is still preliminary, so we do not discuss it further here.

We discuss heuristics such as the greedy heuristic to show that reasonable data (resource) management is possible with profiles. Of course, in practice, data management must contend with multiple profiles and not just one. Below, we discuss a **profile combination** strategies that maps an \(n\)-profile RAP problem to a single profile RAP problem.

### 4.4 Profile Combination

The greedy algorithm for RAP is impractical as given, because it assumes that a single profile determines how a given resource gets allocated. In practice, a given resource may have competing applications, each with its own data requirements specified with a profile. How then, do we adapt this algorithm to handle \(n\) profiles specifying the data requirements for a community of competing applications?

One approach is to merge the \(n\) profiles into one **superprofile** that represents them all. A simple way to accomplish this with two profiles is to append the domain set and utility set equations of one profile, to the domain set and utility set equations of the other (while performing some renaming to ensure that the same domain set name is not used to name two different domain sets). Merging
n profiles simply requires \( n - 1 \) binary merges. Further, if the \( n \) profiles belong to the same profile language (i.e., the same instantiation of the profile language framework presented in Section 3), then the resulting profile is a valid profile in the same language and can be used as input to the greedy algorithm presented in Section 4.2. Profile merging requires some way to normalize utility values across profiles so that profiles are equally represented when merged. But there are many statistical techniques available to perform such normalization (e.g., z scores [MA98]), and therefore we assume profiles have already been normalized for the purpose of this discussion.

As \( n \) grows, the profile that results from merging may become excessively large. One way to cope with this complexity is to recognize when two profiles have defined equivalent domain sets, and to represent these sets once rather than twice in the merged profile. Figure 6 illustrates this process. The two profiles being merged (\( P_1 \) and \( P_2 \)) have domain sets \( D_{n-k}, \ldots, D_n \) in common. The result of merging is the profile, \( P_3 \) with the following characteristics:

- the Domain clause of \( P_3 \) specifies domain sets, \( D_1, \ldots, D_{n+m} \) as defined as in their original profiles.
- the Utility clause of \( P_3 \) consists of three parts:

1. Utility equations for domain sets, \( D_1, \ldots, D_{n-k-1} \), are as in profile \( P_1 \).
2. Utility equations for domain sets, \( D_{n+1}, \ldots, D_{n+m} \) are as in profile \( P_2 \).
3. For every domain set, \( D_i \in \{ D_{n-k}, \ldots, D_n \} \) such that \( (U(D_i) = u_1) \) is the utility equation for \( D_i \) in \( P_1 \), and \( (U(D_i) = u_2) \) is the utility equation for \( D_i \) in \( P_2 \), \( P_3 \) contains the utility equation,

\[ U(D_i) = C(u_1, u_2), \]

such that \( C \) is the combination function defined over pairs of utility expressions in Table 2.

Before illustrating profile merging and utility expression combination with an example, we should point out that the recognition of equivalent domain sets is not required from the perspective
| $i_1$ | $i_2$ | UPTO ($k_2, u_2, u'_2$) | IF $c_2$ THEN $u_2$ ELSE $u'_2$
|-----|-----|----------------|----------------
| $V_1$ | $V_2$ | $V_3$ | IF $c_1$ THEN $u_1$ ELSE $u'_1$

such that:

\[
V_1 = i_1 + i_2
\]

\[
V_2 = \text{UPTO} (k_2, C (i_1, u_2), C (i_1, u'_2))
\]

\[
V_3 = \text{IF} c_2 \text{ THEN } C (i_1, u_2) \text{ ELSE } C (i_1, u'_2)
\]

\[
V_4 = \begin{cases} 
\text{UPTO} (k_1, C (u_1, u_2), \text{UPTO} (k_2 - k_1, C (u_2, u'_1), u'_2)) & \text{if } k_2 > k_1 \\
\text{UPTO} (k_2, C (u_1, u_2), \text{UPTO} (k_1 - k_2, C (u_1, u'_2), u'_1)) & \text{otherwise}
\end{cases}
\]

\[
V_5 = \text{IF } c_2 \text{ THEN } C (u_2, \text{UPTO} (k_1, u_1, u'_1)) \text{ ELSE } C (u'_2, \text{UPTO} (k_1, u_1, u'_1))
\]

\[
V_6 = \begin{cases} 
\text{IF } c_1 \text{ AND } c_2 \text{ THEN } C (u_1, u_2)
\end{cases}
\]

\[
\text{ELSE (IF NOT (c_1) AND c_2 THEN } C (u_1, u'_2)
\]

\[
\text{ELSE (IF NOT (c_1) AND c_2 THEN } C (u'_1, u_2)
\]

\[
\text{ELSE C (u'_1, u'_2)))}
\]

Table 2: C: The Combination function for profile utility expressions

of the correctness of our merging technique. The semantics of profiles is such that a given data object can belong to multiple domains, and will have a value that is the sum of values it takes from each of these domains, and this accumulated value is what is considered by the greedy algorithm described in Section 4.2. But the degree to which equivalent domain set expressions can be recognized will affect the size of merged profiles. Therefore, our strategy is to recognize equivalent domain set expressions as often as we can. Of course, depending on the language for expressing domain set expressions, this task may or may not be straightforward. If, for example, domain sets are specified with URL's, then equivalent domain sets are simply those specified with the same URL's.\(^5\) Once domain sets are expressed with queries, domain set equivalence becomes undecidable. We are presently considering strategies for using semantic information to determine query equivalence in special cases, but this largely remains future work and is discussed no further here.

An Example: We trace profile merging with an example, merging the Traveler profile of Figure 2b to a similar Traveler profile (NYTraveler) defined for visitors to NYC, and shown in Figure 7a. The result of adding these profiles is shown in Figure 7b. Note the the utility expression for the domain shared by Traveler and NYTraveler is derived by the rules of Table 2, as shown by the derivation below.

\(^5\) Though, URL's of mirror sites would not be recognized as equivalent in this approach.
PROFILE NYTraveler

DOMAIN
R = related:www.hertz.com
S = JFK Airport AND downtown AND "shuttle schedule"
D = "Downtown Manhattan" AND hotel AND "directions from JFK Airport"

UTILITY
U (R) = UPTO (3, 1, 0);
U (S) = UPTO (1, 3, 0);
U (D) = UPTO (1, 2, UPTO (1, 1, 1))

END

(b)

PROFILE Merge (Traveler, NYTraveler)

DOMAIN
R = related:www.hertz.com
S = Logan-Airport AND downtown AND "shuttle schedule"
D = "Downtown Boston" AND hotel AND "directions from Logan Airport"
S = JFK Airport AND downtown AND "shuttle schedule"
D = "Downtown Manhattan" AND hotel AND "directions from JFK Airport"

UTILITY
U (R) = UPTO (2, IF (#D > 0) THEN 2 ELSE 1, UPTO (1, 1, 0));
U (S) = UPTO (1, 2, 0);
U (D) = UPTO (1, 3, 0);
U (D) = UPTO (1, 2, UPTO (1, 1, 1))

END

(b)

Figure 7: Profile NYTraveler (a) and Merge (Traveler, NYTraveler) (b)

\[ C \left( \text{UPTO} \left( 2, \text{IF} \left( \#D > 0 \right) \text{THEN} 1 \text{ELSE} 0, 0 \right) \right) = \text{UPTO} \left( 2, \text{IF} \left( \#D > 0 \right) \text{THEN} 1 \text{ELSE} 0, 1 \right) \]
\[ C \left( \text{UPTO} \left( 2, \text{IF} \left( \#D > 0 \right) \text{THEN} 1 \text{ELSE} 0, 0 \right) \right) = \text{UPTO} \left( 2, \text{IF} \left( \#D > 0 \right) \text{THEN} 1 \text{ELSE} 0, 1 \right) \]

Below we consider the value of a specific set of objects, O, with respect to the profile resulting from merging Traveler and NYTraveler. The point of this example is to illustrate that merging produces the intuitive result — that the result of merging two profiles, P1 and P2, is a profile whose value for O is the sum of the values for O taken by profiles P1 and P2 separately. A formal proof showing that

\[ \| \text{merge} (P1, P2) \| (O) = (\| P1 \| (O)) + (\| P2 \| (O)). \]

for all P1, P2 and O can be shown by structural induction on O. We omit the proof in the interest of space.
For this example, let \( O = \{ r_1, r_2, r_3, db_1, dny_1, dny_2 \} \) such that:

\[
\begin{align*}
\sigma_R &= \{ r_1, r_2, r_3 \} \\
\sigma_S &= \emptyset \\
\sigma_{S_2} &= \emptyset \\
\sigma_D &= \{ db_1 \}, \text{ and} \\
\sigma_{D_2} &= \{ dny_1, dny_2 \}
\end{align*}
\]

The reader can use the semantic equations of Section 1 to verify that \( \llbracket \text{Traveler} \rrbracket (O) = 3 \) and \( \llbracket \text{NYTraveler} \rrbracket (O) = 6 \). Consider now, the value \( O \) with respect to \( \text{Merge} \) (Traveler, NYTraveler) (Figure 2b):

\[
\llbracket \text{Merge} \text{ (Traveler, NYTraveler)} \rrbracket (O) = (\llbracket UT_R \rrbracket (\sigma_R(O), O)) + (\llbracket UT_S \rrbracket (\sigma_S(O), O)) + (\llbracket UT_{S_2} \rrbracket (\sigma_{S_2}(O), O)) + (\llbracket UT_D \rrbracket (\sigma_D(O), O)) + (\llbracket UT_{D_2} \rrbracket (\sigma_{D_2}(O), O))
\]

such that

\[
\begin{align*}
\llbracket UT_R \rrbracket (\sigma_R(O), O) &= \llbracket \text{UPTO} (1, 2, 0) \rrbracket (\emptyset, O) = 1 \cdot 0 = 0 \\
\llbracket UT_{S_2} \rrbracket (\sigma_{S_2}(O), O) &= \llbracket \text{UPTO} (1, 3, 0) \rrbracket (\emptyset, O) = 1 \cdot 0 = 0 \\
\llbracket UT_D \rrbracket (\sigma_D(O), O) &= \llbracket \text{UPTO} (1, 1, 0) \rrbracket (\{ db_1 \}, O) = 1 \cdot 1 = 1 \\
\llbracket UT_{D_2} \rrbracket (\sigma_{D_2}(O), O) &= \llbracket \text{UPTO} (1, 2, \text{UPTO} (1, 1, 1)) \rrbracket (\{ dny_1, dny_2 \}, O) \\
&= \llbracket 2 \rrbracket (\{ dny_1 \}, O) + \llbracket \text{UPTO} (1, 1, 1) \rrbracket (\{ dny_2 \}, O) \\
&= 2 + 1 = 3 \text{ and}
\end{align*}
\]

\[
\begin{align*}
\llbracket UT_R \rrbracket (\sigma_R(O), O) &= \llbracket \text{UPTO} (2, \text{IF} (#D > 0) \text{ THEN} 2 \text{ ELSE} 1, \text{UPTO} (1, 1, 0)) \rrbracket (\{ r_1, r_2, r_3 \}, O) \\
&= (\llbracket \text{IF} (#D > 0) \text{ THEN} 2 \text{ ELSE} 1 \rrbracket (\{ r_1, r_2 \}, O)) + (\llbracket \text{UPTO} (1, 1, 0) \rrbracket (\{ r_3 \}, O)) \\
&= (\llbracket 2 \rrbracket (\{ r_1, r_2 \}, O)) + (\llbracket 1 \rrbracket (\{ r_3 \}, O)) \\
&= 4 + 1 = 5
\end{align*}
\]

Thus,

\[
\llbracket \text{Traveler} + \text{NYTraveler} \rrbracket (O) = 9 = (\llbracket \text{Traveler} \rrbracket (O)) + (\llbracket \text{NYTraveler} \rrbracket (O)).
\]

5 Related Work

User profiles form the basis of many types of information delivery systems, ranging from information filtering applications to the personalization of content on the World Wide Web. Our notion of profiles is unique in that it combines a language for specifying predicates over data items with detailed specifications of the user’s preferences, priorities, and requirements. Still, there has been significant work in user profile modeling and management upon which we can draw.

User profiles for Web-based applications (such as MyYahoo, PointCast [RD98], etc.) are typically fairly simple, allowing the user to specify particular categories of information that they are interested in receiving. Such categories are often referred to as channels, thereby emphasizing the similarity of the service to that of broadcast media. This approach to building information dissemination systems is generally known as the publish/subscribe model [OPSS93]. Publish/subscribe
systems tend to use a *walled garden* approach, in which the universe of data that can be delivered to the user is restricted to specific content sites. Most systems allow simple, channel-specific predicates to be applied to the data on the channels selected by the user, for example, to specify particular companies (for stock prices), cities (for weather), or teams (for sports scores) of interest to the user. The Grand Central Station system [LEF98], developed at IBM Almaden provides a more general form of predicates over its channels, and is therefore closer to our notion of the domain specification portion of a profile.

User profiles for text-based data have been extensively investigated in the context of Information-Filtering and Selective Dissemination of Information research [FD92]. The systems in these areas use techniques from the Information Retrieval (IR) world for filtering unstructured text-based documents [BC92]. In general, IR profile systems use either a Boolean model or a similarity based model. In the Boolean model a user profile is constructed by combining keywords with boolean operators (e.g., And, Or, Not), and an “exact match” semantics is used — a document either satisfies the predicate or not. Similarity-based models use a “fuzzy match” semantics in which the profiles and documents are assigned a similarity value (typically based on a Vector Space Model [?]). A document whose similarity to a profile is above a certain threshold is said to match that profile. The Stanford Information Filtering Tool (SIFT) [YGM99], is a well-known content-based text filtering system for Internet News articles. With the advent of XML, filtering of web-documents based on structure as well as content has become more feasible. The XFilter system [AF00] is a recent example of such a filtering system.

Our notion of user profiles is also related to the notion of Continuous Queries (CQ). Continuous Queries are standing queries that allow users to be informed when updates of interest occur in a database. Early work on CQ for relational databases was done by Terry et al. [TGNO92]. More recently, several CQ systems have been proposed for information delivery on the Internet. OpenCQ [LPT99] employs an SQL-like query language and runs on top of a distributed information mediation system which integrates heterogeneous data sources. The NiagaraCQ system [CDTW00b], allows the specification of standing queries using an XML-based query language. The C3 project [CAY98] provides a service that allows users to subscribe to changes in semi-structured information sources.

It was mentioned earlier that we use profiles to facilitate automated data management. Other work [CN97, CN98] has looked at ways to automate in part the DBA function. This work does not consider the use of profiles, but instead provides a way to use a representative workload to test a system’s responsiveness under this load. It also focuses on the tuning of a standard DBMS, while we extend the notion to a network setting.

Data freshening has been studied before from the synching operation of the Palm Pilot to recent work on synchronizing local copies of a database from Stanford [CM00b]. This later work studies freshening policies based on how often the underlying data elements change. In our work, we refresh data items on the basis of their relative importance as specified in profiles. Combining profiles and their utility specification is related to [AW00]. That work defines a generalized operator for combining preferences, but it does not address the problem of determining the utility of a set of objects.

Finally, our data recharging application will make heavy use of techniques to generate and deliver data at multiple resolutions. Multi-resolution delivery of media has long been advocated for mobile environments (e.g., [Kat94]). We plan to extend these techniques to deal with structured and semi-structured documents as well as to aggregated data delivered from databases.
6 Conclusions

This paper introduced the idea that in a network setting it is possible to base data management decisions on profiles collected from the user community. To this end, we introduced a profile language framework along with its formal specification. We further discussed some issues regarding the processing of these profiles in pursuit of two sample applications: data recharging and data freshening.

We view this work as a migration of data management ideas and techniques into the wide-area network setting. This approach could have an enormous impact on the performance of the web as an integrated information service. We believe that profiles, as described in this paper, are an enabling technology that could spawn a rich new area of research in network data management. The use of declaratively specified profiles to drive data management policy promises benefits similar to the profound effect that declarative queries had on database system technology. This paper is a first step in that direction.

References


