A Web Search-centric Approach to Recommender Systems with URLs as Minimal User Contexts

W. K. Chan[1]
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

Yuen Yau Chiu
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
dbskings@gmail.com

Yuen Tak Yu
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
csytyu@cityu.edu.hk

Abstract

In service-oriented computing, a recommender system can be wrapped as a web service with machine-readable interface. However, owing to the cross-organizational privacy issue, the internal dataset of an organization is seldom exposed to external services. In this paper, we propose a higher level recommender strategy INSERT that guides the underlying external universal recommender to suggest a set of indexes. INSERT then matches the title of each top-ranked index entry with the domain-specific keywords in the organization's internal dataset, and further directs the universal recommender to verify the popularity of such matching. INSERT finally makes recommendation based on the verification results. INSERT also employs URLs taken from a client as user contexts, which is challenging because URLs contain little content. Our experiment shows that this strategy is feasible and effective.

Keywords: recommender system, in-store shopping, URL, matching, implicit user profile.

1. Introduction

A recommender system [3] is a class of applications that ranks a set of available choices based on some given criteria [55] and available set of inputs. Example inputs include data available on mobile clients, direct user inputs, and data on various websites. Criteria are often derived from similarity coefficients such as how close a choice to an input is. Recommender systems have the potentials to support activities in software development and maintenance projects [23][41], such as code reuse [41] and semi-automated fault localization in faulty programs [28][31][52][58][59] or web services [26]. Moreover, if the service-oriented paradigm to software development is adopted, developers may use external services to implement a particular application task or software development task, which effectively re-formulates an aspect of a traditional software development problem into a service selection problem [57]. For one example, if a restaurant recommendation service used by an application is faulty, rather than performing corrective maintenance on this service for the application, developers may substitute it with another restaurant recommendation service. For another example, a test case prioritization service may select among prioritization metric services and bind to one of them to conduct service-based regression testing with a particular prioritization goal [33]. Researchers tend to consider that such a strategy encourages reuse of service-based components [45][53].

In previous work [3][55], research challenges in the quality aspect of inputs have been studied, such as obtaining a minimal set of important, accurate, and implicit user preferences. To the best of our knowledge, many existing researches on recommender systems focus on improving the algorithm of a recommender system or enhancing the input quality or the item quality with profiles or ontology. They have not studied from the perspectives of software architecture [44][47] and service usage adequately.

In service-oriented computing, a recommender system can be wrapped as a web service with machine-readable interface. A service-based application may discover and bind to such recommender services to

[1] Correspondence author.

implement some of its own functionalities. However, owing to the cross-organizational privacy issue [56], the internal dataset of an organization is seldom exposed to external services. This issue poses a technical challenge on the use of an external recommender service that suggests choices based on a corporate's internal dataset. In this paper, we address this problem.

In terms of software architecture [7], an application may have multiple and geo-distributed components. Prior researches such as [54][55] have recognized that if a remote component (specifically, a mobile client) with respect to the main component of a recommender system is involved, the amount of data and when they are used are important factors in designing a fast and effective recommender system. It would be too simplistic to assume that such a main component must be a centralized system. This paper considers a recommender application composed of at least three distributed components: a mobile client, an external service that stands for a universal recommender, and a high level recommender whose recommendation capability is derived from the universal recommender.[2] We study whether a derived (aka high level) recommender can be effectively built on top of a universal recommender. The challenge is that the universal recommender is insensitive to the contexts of individual special-purpose applications and, hence, the information directly chosen by such a recommender is often imprecise.

In terms of service usage, some researchers such as [14] have recognized the use of URLs on blogs to analyze user preferences[3], but they have not recognized that analysis or indexing of these blog contents could be provided by a third-party service. Using a third-party service helps leverage the analysis efforts and costs among the clients of the same service. A naïve strategy is to pass all the URLs from the mobile clients as well as all the information about all the items kept by a high level recommender to a universal recommender, and then the latter does the matching and ranking of the suggested items. Such a strategy, however, fails to address the cross-organization privacy issue. A key challenge of the research is how to reduce the communication overhead while enabling a high quality recommendation list produced to the user.

In this paper, we propose a strategy called **IN**ternet **SE**arch-based **R**ecommender **T**echnique (**INSERT**). It derives a high level recommender based on a universal recommender which is realized as a widely-used web search engine in practice. Specifically, INSERT uses the URLs kept by the web browser of a smart phone as *pointers*[4], a universal recommender to determine which URLs are specific, and the set of returned index entries of those specific URLs as the user profile. INSERT further pairs up the ranked index entries returned by the universal recommender and its own information about the title of each index entry, and requests the underlying universal recommender to estimate the popularity of every such match. Finally, INSERT constructs a recommendation list in descending order of the number of matched results estimated by universal recommender.

INSERT has a number of unique properties. First, by building on a popular universal recommender, the technological upgrade of the basic recommendation capability is encapsulated within the universal recommender. Such delegation of responsibility would be attractive to small software developers or individuals who do not have the resources to implement, operate and maintain all the technological issues managed by universal recommenders. Second, INSERT uses the ranking of the pages provided by universal recommenders as the basic criterion to order the information to be recommended to the clients. Interestingly, once an URL is available from a web browser, INSERT can compute a recommendation list without visiting the website pointed by the URL or fetching its page contents. Last, but not the least, INSERT uses the

---

[2] The situation is similar to adopting a service composition approach to derive high level functionalities based on a set of more primitive services.

[3] Many location-aware recommender systems assume that the location of a mobile client or a user is available, but this is untrue if a user chooses to disable such a location estimation feature.

[4] When such pointers are not available prior to activating INSERT, the user may simply browse the items they want first and then re-activate INSERT later.

information (i.e., URLs) readily available from web browsers of smart phones. The use of such a minimal amount of information to personalize the recommendation list for customers is challenging, yet our study demonstrates that this idea is feasible and fruitful.

The main contribution of this paper is threefold. (1) To our knowledge, this paper proposes the first model that uses URLs as pointers to user profiles and an external universal recommender to summarize URLs that refine the implicit user profiles by progressively exploring the contents returned by the universal recommender. In so doing, there is no need to revisit the content pointed by any URL directly. (2) It proposes a new strategy that derives recommendation capability based on the notion of service composition. We show that our proposed strategy is feasible and the result is better than directly using the universal recommender. (3) It proposes an architecture that leverages the Internet and a real-life web search engine as the source of recommendation matching. It provides a viable solution to small developers and end-users.

The rest of this paper is organized as follows. We first describe a motivating example in Section 2, followed by introducing the fundamental concepts related to INSERT in Section 3. We then describe INSERT in detail in Section 4. Section 5 reports an evaluation of INSERT and discusses issues related to its implementation and application. Section 6 reviews related work and compare it with ours. Finally, Section 7 concludes the paper.

2. Motivating Example

In this section, we describe an example to motivate the need for our proposed strategy. When presenting our ideas in this paper, we shall refer to two fictitious brands, SANY and Nikan, instead of naming real commercial brands.

Suppose that a customer wants to buy a digital camera. For this purpose, the customer searches the Internet and browses the camera's product description web pages, which possibly reside in a web site different from that of the shop from which he/she intends to buy the camera. After doing some surveys over the Internet, the customer then visits a shop (in person) which sells the camera.

Suppose further that the customer simply provides a keyword such as "digital camera" to the recommender system of a shop that sells SANY products only. Such a keyword-driven strategy uses the *explicit* information provided by the customer as well as the attributes of the product for matching. Owing to the imprecision of the provided keywords to describe the customer's intent, this strategy may recommend information items that are not specifically relevant to the profile of the customer. For instance, although the customer may know from the web pages of the shop that it sells SANY products only, the shop may actually sell three different models of SANY digital camera, say, DSLR BB500, DSLR BB550, and DSLR BB850. Suppose that the customer actually only wants to buy a camera of model number SANY DSLR BB850 (or BB850 for short). The presence of the other two related but unwanted models (BB500 and BB550) render the recommendation list imprecise for the customer. For instance, one possible order of the three models recommended by the system is ⟨BB550, BB500, BB850⟩. This ordered list may confuse or distract the customer, as the target camera model is placed at the end of the list after two unwanted models.

One way to improve the quality of the shop's recommendation list is to use the method of *tag co-occurrence* [51]. This method essentially computes the percentage of the tags (or attributes) associated with an item in common with the inputted keyword, and recommends those items with a higher percentage first. Although this method may provide an improved permutation of the recommendation list, the less relevant or irrelevant items remain in the list. The actual permutation depends on the popularity of tags associated with each item, rather than taking into account this particular customer's need. Using a location-aware recommender system does not help in this scenario because the customer is already *in* the shop. A location-aware system may simply recommend a list of *all* matched cameras in the shop according to the location

information. This essentially requires the customer to manually resolve the imprecise recommendation list produced by the system, which has no information to estimate what camera(s) he/she actually wants.

Another way to obtain a more precise recommendation list is to require the customer to provide more precise keywords, such as memorizing or recording down the exact model number (i.e., BB850 in this example) followed by recalling and providing the keyword explicitly. We consider this solution unsatisfactory, and propose improvements in two aspects. First, to relieve the customer's burden, the recommender may use an alternate strategy, namely, to identify the characteristics of the customer through collecting an implicit user profile. Secondly, to relieve the burden of the recommender system, some lower level recommendation activities may be delegated to another recommender. We are not aware of public literature that investigates this architectural option. In Section 4, we shall present the details of our integrated strategy that takes both of these two basic ideas into account.

After being satisfied with the camera's product description, the customer may proceed with either online shopping or in-store shopping [4]. *Online shopping* (OS) refers to purchases through the Internet, for example, by placing the order and paying on the shop's web site, and then waiting for the physical delivery of the product to the specified address. This is convenient for the customer, who can simply shop at home without travelling to the shop. Nonetheless, OS is known to have certain weaknesses [27], such as what one buys may not be exactly what one orders [9][50], failure to obtain immediate assistance in case of problems [11][16][27], and an indefinite length of time between the placing of order and delivery of the product.

In this paper, we consider the alternative scenario in which the customer performs *in-store shopping* (ISS). That is, the customer pays visit to a shop in person and buys the product there after having searched for the desired product through the Internet. In so doing, the customer may physically examine the product in the shop and obtain it immediately after payment, but would need to spend time and effort not only in travel, but also in locating the desired product within the shop. In particular, when the shop is popular and crowded, the salespeople may be unable to provide personalized recommendations to the customer or locate the desired product for him/her. To address these problems, we are going to propose a strategy for implementing a specialized, efficient and fully-automated recommendation system, which provides a personalized recommendation list to the customer who is performing ISS. Our strategy, INSERT, does so by taking advantages of the web URLs stored in the customer's mobile device (such as a smart phone), an underlying freely available universal recommender (which is realized as a public generic web search engine), as well as the shop's database of its product information (which we call the *corporate dataset* or the *content dataset*). We shall refer to the ISS scenario described here as a running example to illustrate our work in this paper.

3. Preliminaries

Before describing the details of our proposed strategy, it is necessary to introduce some preliminary concepts and terminologies in this section first.

3.1 URL and page content

Figure 1 depicts an ISS scenario of using a recommender system which implements our proposed strategy, INSERT. At the top left corner of Figure 1, a user surfs the Internet via a smart phone. In the illustrated scenario, the user accesses four URLs, namely, url_1, url_2, url_3, and url_4. A ***URL*** is a string (such as http://www.cs.cityu.edu.hk/) which acts as an address or pointer to a web location (usually a web page).

The ***page content*** pointed by a URL is the source content of a page displayed on a web browser (e.g., Firefox). Technically, the page content can be a piece of text, typically written in a markup language such as HTML or XML. Alternatively, it can also be a piece of multimedia item (such as image or video), an operation of a web service, and so on. Figure 1 illustrates the contents of the pages pointed by the four URLs (url_1, url_2, url_3, and url_4), which correspond to descriptions of two digital cameras (SANY DSLR BB850 and Nikan DSLR BBA50), the portal information of a university department (CityU CS Homepage), an e-coupon (Restaurant A's Coupon), and a television (SANY TV NX8000), respectively.

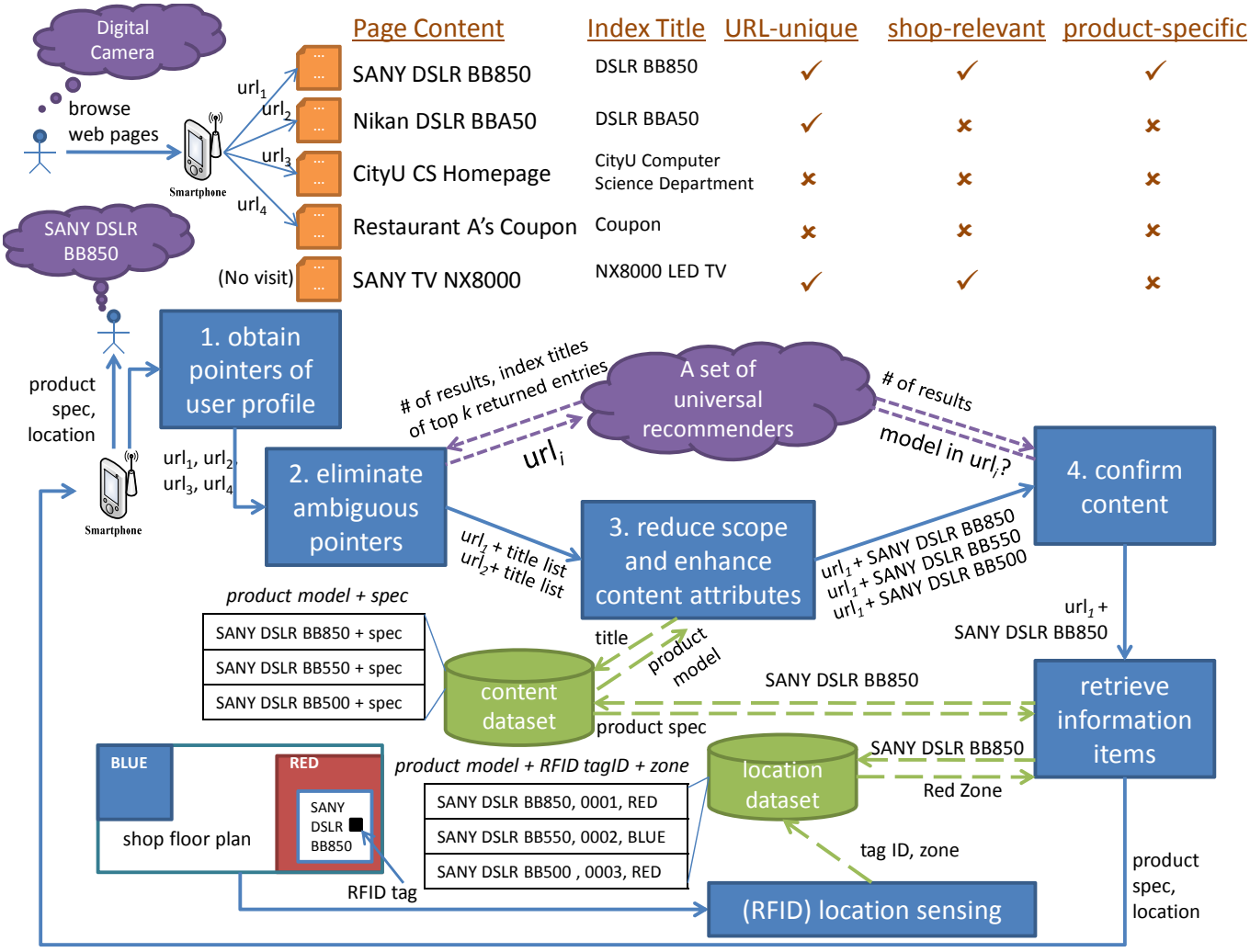

Figure 1. A scenario of using our recommender system in the context of in-store shopping. The illustrations from top left and then clockwise are:

(a) A user who visits a few web pages using a smart phone.

(b) A list of attribute values of the web pages relevant to our motivating example.

(c) A workflow scenario (steps 1–4) that illustrates how our recommender system generates an accurate, on-the-fly, and personalized recommendation list of products based on the minimal information available on a typical smart phone, the ubiquitous information available on a web search engine (which serves as a universal recommender), and the up-to-date locations of the products in the shop.

3.2 Index title

After a user keys in a string of keyword, a typical web search engine will return a sequence of web pages. On each returned page, there is a list of index entries. Usually, each index entry will consist of several lines of text and probably some other information. One (usually the first) of these lines contains what we call the ***index title*** of the entry, which is the text that describes the main information of the index entry.

For instance, by typing the keyword "CityU CS", the first entry returned by Google [20] consists of four consecutive lines, namely, "Department of Computer Science - CityU CS" (with an embedded hyperlink that points to http://www.cs.cityu.edu.hk/), "Department of Computer Science Department of Computer Science - City University of Hong", "Kong.", and "www.cs.cityu.edu.hk/".

In the above example, the first line is the index title, the second and third lines together contain the elaborated details of the index title, and the fourth line is the URL that links to a web page. Figure 2 depicts a simplified structure of a sequence of web pages returned by Google so that readers can have a clearer picture of how our strategy works as it is described later.

Owing to the popularity of Google [20] worldwide and Google Hong Kong [21] in our region, throughout this paper, we shall use these two web search engines to illustrate the implementation of universal recommenders employed by INSERT. Nevertheless, INSERT is not bound to Google, and other web search engines may be used instead (such as Baidu [6], a popular web search engine in China).

3.3 Contexts in pervasive computing

Context-aware programs in pervasive computing interact with their environments via environmental attributes known as ***contexts*** [2][10][43][52]. A software application may make exclusive use of the contexts obtained from the underlying middleware [10][42][52] to optimize the computation or results to fit for the environment represented by the contexts [52]. The application may also listen to various context-aware events and invoke self-adaptive actions if needed [46]. These features enable the application to be aware of the environmental feedback on its behavior [36]. Context-awareness is a strategic feature in context-aware computing that enables an application to be intelligent. A recommender system can be considered as a kind of context-aware application.

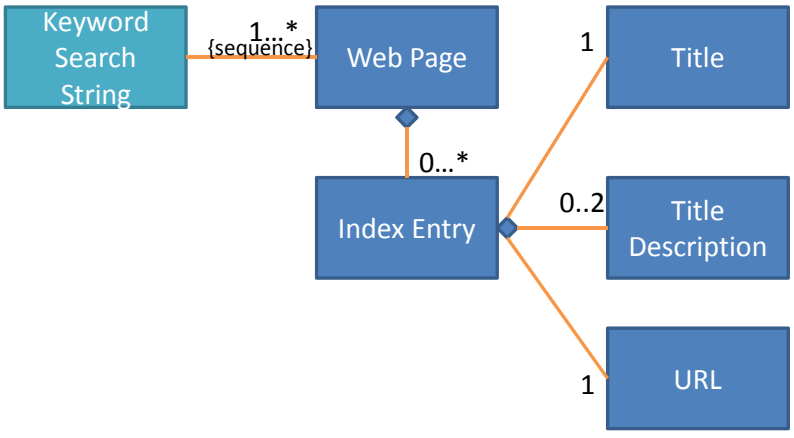

Figure 2. Sample structure of a web page returned by a web search engine and its relation with the input to the web search engine

4. The Proposed Strategy: INSERT

In this section, we present INSERT, which consists of four workflow steps organized in a dataflow pipeline architecture [7] as shown in Figure 1.

Step 1: Obtain pointers of user profile

We observe that many users surf the Internet by using web browsers on mobile devices such as smart phones or PDAs. Moreover, many web browsers (such as Firefox, Internet Explorer, Chrome, or Safari) by default keep the visited URLs of the browsed web pages automatically. The web pages pointed by the visited URLs would provide valuable indicative information on the customer's choices and preferences. To capture this intuition, INSERT uses the customer's URL history as the *starting point* of obtaining the implicit user profile.

Formally, a URL is a string p, which serves as a pointer (or address) to a web location. The URLs kept by a web browser (on a smart phone) can be ordered by their timestamps, which can be modeled as a sequence $P = \langle p_1, p_2, \ldots, p_{|P|} \rangle$. For instance, the smart phone in Figure 1 has kept the sequence $\langle \text{url}_1, \text{url}_2, \text{url}_3, \text{url}_4 \rangle$.

Step 2: Eliminate ambiguous pointers

The pointers in P may be imprecise. For instance, the page contents for url_3 and url_4 are not about digital cameras at all. Moreover, a URL link can be broken or the page it points at can be temporarily unavailable, removed, or modified. Some pointers may refer to generic catalogs or pages generated dynamically by the AJAX functions of some web pages. Furthermore, some pages may contain dynamic multimedia contents, such as streaming videos. Therefore, a re-visit or analysis of these web pages directly may be tedious or futile. An in-house (probably domain-specific) recommender cannot easily predict these scenarios and address them cost-effectively.

Rather than using an explicit ontology to analyze these pointers, INSERT delegates the analysis to a universal recommender, which may simply be realized by a public and widely-used web search engine, such as Google [20] or Baidu [6]. As such, we shall use the terms *universal recommender* and *web search engine* interchangeably in this paper.

To ease our presentation, we model a universal recommender as a mapping ***WSE*** that maps an input to a sequence of index entries, whose structure is shown in Figure 2. Furthermore, for any pointer p, we denote the index title of the entry at position i of the returned sequence of entries by $WSE(p)[i].title$.

Our insight in this step can be illustrated as follows. When a user provides a sensible keyword (e.g., "Digital Camera") to a web search engine, the latter tends to return a large number of estimated "hits". For instance, to our understanding, the name of our department is unique enough. However, when we performed a trial search with the keyword "Hong Kong CityU CS" as input, Google estimated that there were 4,040,000 results. These results included many web pages containing the information about *all* CS departments in Hong Kong, as well as those containing our department's prospectus and course information. On the other hand, when we searched by the URL "www.cs.cityu.edu.hk" as input, Google still returned 433 results. We tried again using examples drawn from different categories, and the results are as shown in Table 1. Our insight is that using a pointer (a specific URL) as input for searching is far more precise than using a keyword. An end-user may directly use URLs to identify many web pages, but these pages may not be related to the choices available in a shop. (We model the set of all choices available in a shop as a content dataset in Figure 1).

In this step, INSERT tries to reduce the number of pointers (URLs) to be considered in the subsequent steps. Two criteria can be used for this purpose. First, a pointer can be discarded if searching by it returns no index entry by the web search engine, since there is insufficient evidence that the URL is related to an item

Table 1. The number of returned results for different kinds of inputs by a web search engine (Google)

Concept Category	**Concept (Keyword)**	**Search by Keyword**	**Search by Website URL (A)**	**Search by Concept Homepage URL (B)**	**(B) ÷ (A)**	**URLs**
Product	Sany DSLR-A850	139,000	184	**2**	0.011	Website: http://www.sonystyle.com.hk Concept homepage: http://www.sonystyle.com.hk/ss/product/alpha/a850/index.jsp
Education Degree	CityU CS Bachelor of Computer Science	515,000	433	**6**	0.013	Website: http://www.cs.cityu.edu.hk/ Concept homepage: http://www.cs.cityu.edu.hk/academic/bsccs
IEEE Conference	ICSE 2010	888,000	2,860,000	**1**	< 0.001	Website: http://www.ieee.org Concept homepage: http://www.ieee.org/conferences_events/conferences/conferencedetails/index.html?Conf_ID=16433
Person	Wen Jia Bo	1,210,000	30,600,000	**26**	< 0.001	Website: http://en.wikipedia.org Concept homepage: http://en.wikipedia.org/wiki/Wen_Jiabao

that the user wants. (We note that directly matching a pointer with the content dataset is impractical owing to the lack of concrete data in the pointer string. For instance, every DOI of a paper is a string without the title and content of the paper.) Second, if searching by a particular pointer results in a large number of returned index entries by the web search engine, the pointer is considered not precise enough. Depending on the search domain, an expert may therefore preset a positive integer, α, as a threshold for eliminating these pointers when the number of returned index entries exceeds α. In summary, INSERT removes a pointer p in P whenever $|\mathrm{WSE}(p)| = 0 \vee |\mathrm{WSE}(p)| > \alpha$, where $|\mathrm{WSE}(p)|$ denotes the number of index entries returned by the web search engine given the input p.

Consider again our example as depicted in Figure 1. After this step, only two URLs are returned, that is, $P = \langle \mathrm{url}_1, \mathrm{url}_2 \rangle$. Note, however, that url_2 is not the URL of a web page about the target digital camera SANY DSLR BB850. In the next two steps, INSERT eliminates these less relevant URLs by means of even better and more restricted scope of matching.

Step 3: Reduce scope and enhance content attributes

In this step, INSERT further enhances every pointer p with information from the shop's corporate (content) dataset governed by a high level recommender. Note that for each p in P obtained in Step 2, INSERT has already successfully located a sequence of index titles from the universal recommender. We observe that every such index title is a matched concept of the corresponding index entry. INSERT uses this matched concept for full-text string matching with the attributes of items kept in the content dataset of the high level recommender. For instance, in Figure 1, the content dataset contains no Nikan item, and thus there is no match returned by the title for url_2. On the other hand, the content dataset contains information of a few items in the form "SANY DSLR BB*0", where the wildcard "*" represents "85", "55", or "50". Hence, the recommender returns the three tuples (SANY DSLR BB850), (SANY DSLR BB550), (SANY DSLR BB000), each containing a SANY DSLR product model.

To ease our reference, we model the matching operation in this step as a function f that maps an index title t to a sequence of tuples, each tuple being a sequence of attribute values of an information item. For

instance, suppose that there is only one index title for url_1. Then $f(WSE(\text{url}_1)[1].title)$ = ⟨(SANY DSLR BB850), (SANY DSLR BB550), (SANY DSLR BB000)⟩. We shall refer to the three tuples in the sequence as $f(WSE(\text{url}_1)[1].title)[j]$, j = 1, 2, and 3, respectively.

INSERT now contains more information about each user profile element, namely, an URL and a combination of title and attribute values of the corresponding items. In the above example, if the current result were used as the final output recommendation list, the accuracy is only 33.3%. In the next step, INSERT further eliminates those probably unwanted items from this list.

Step 4: Confirm content

In this step, INSERT confirms whether the currently identified information is really contained in the content pointed by the pointers of user profile.

It might appear that we can simply proceed by checking that the attribute values associated with each title (see Step 3) are on the page pointed by the URL that identifies the title (i.e., url_1 for the running example). However, as mentioned before, the contents of such pages may be videos, audio clips, or documents encoded in various formats. Checking such pages directly can be both inefficient and ineffective. Instead, we observe that a real-life widely-used web search engine has provided comprehensive indexing on each indexed web page. In fact, these pages passed from Step 3 must have been indexed by the web search engine, or else the URL would have already been suppressed in Step 2.

INSERT uses the containment operator provided by a web search engine (e.g., "<attribute value>" inurl <URL> for Google [20] and Baidu [6]) to confirm whether the page pointed by a user profile pointer contains the required attribute value. For instance, by issuing the following query in Google, *no* result is returned by the web search engine, indicating that the attribute value does not appear in the concept whose web page was browsed by the customer.

"SANY DSLR BB550" inurl http://www.cs.cityu.edu.hk/academic/bsccs

On the other hand, if the URL is http://www.sonystyle.com.hk/ss/product/alpha/a850/index.jsp, two index entries are returned by the web search engine. To ease our reference, we overload the functor *WSE* to accept three parameters (attribute value, "inurl", url) and return a sequence of index entries.

Technically, in this step, INSERT eliminates a tuple T of attributes associated with a title and a pointer p in P whenever $\forall_{a \in T}$ |WSE(a,"*inurl*", p)| = 0. For instance, in our running example shown in Figure 1, the page of url_1 does not contain "BB550", and so |WSE("SANY DSLR BB500","*inurl*", url_1)| = 0. On the other hand, |WSE("SANY DSLR BB850","*inurl*", url_1)| = 1. Therefore, in this example, only one tuple (url_1, $f(WSE(\text{url}_1)[1].title)[1]$, $\max_{a \in T}\{|\text{WSE}(a,\text{"}inurl\text{"}, p)|\}$) is generated. In case multiple tuples are generated, INSERT sorts them in descending order of the value specified in the third element of each tuple. This value represents the estimated number of results that the web search engine successfully matches (via *inurl*) the attribute against the information in the user profile. The greater the estimated number of matched results, the higher confidence there is a relevant match. In the case of a tie, INSERT resolves it randomly.

Readers may wonder whether Step 2 and Step 4 can be combined into one. Note that the starting point of Step 2 is a pointer, which does not have any content information, rendering it infeasible to match the content using the containment operation in Step 4.

5. Evaluation

To evaluate the practicality and efficiency of INSERT, we have implemented it in a recommender prototype system whose client runs on a smart phone. In this section, we first describe the prototype implementation, followed by an empirical comparison of INSERT with three other techniques. Finally, we discuss the lessons learned from the evaluation.

5.1 Implementation of a prototype

As we design a prototype and implement INSERT, we had in mind an application similar to what we describe in the motivating example (Section 2). While INSERT returns a recommendation list of items for the user, it has no information regarding the physical location of the items in the shop. Thus, in our prototype, we enhance INSERT so that the ranked list of recommendation is further used to retrieve the item details kept by the shop's content dataset. In particular, when the requested entity is a physical item (rather than a pure information item), the prototype system will try to direct the user to the place where the item is physically located. One way this can be done is to use RFID location sensing to determine the zone in which the required item resides. This zone information would be helpful even if the location of the item is not exactly reported, particularly when the shop is large or the salespeople are too busy.

For instance, for the tuple (url_1, $f(WSE(\text{url}_1)[1].title)[1]$, $\max_{a \in T}\{|\text{WSE}(a, \text{“url”}, p)|\}$) generated by INSERT, our system can retrieve the product specification of the BB850 model. Moreover, from the location dataset (part of the shop's database), our system can further display the zone in which the item is detected to be present (e.g., RED zone for the BB850 model as shown in Figure 3(e)).

Figure 3 shows the client side of the proof-of-concept prototype of INSERT. The prototype system consists of three major components. The first component is an Android application that runs on top of the Google Android OS (with SDK 1.1). It is responsible for extracting the URLs from the page history of the web browser in the mobile device and sending the URLs to the second component (Step 1 of INSERT). It is also responsible for displaying the final recommendation list on the mobile device. The second component resides in a backend server and is written in Java, which implements all the remaining steps (Steps 2–4) of INSERT. It also implements the retrieval of item details. Note that this backend server has no GUI interface and, hence, we do not show them graphically in this paper. The third one is the RFID location sensing component. We use C# to implement this component, primarily because the available RFID API was written in C#. MySQL is used for implementing the content database. Connector-ODBC 5.1.6 is used to connect the server components and the database. The RFID location sensing component uses Tracient TraceConnect to connect to the RFID readers. This third component is used to report the areas where the products are displayed. All server-side programs are deployed on Windows Server 2008 running on a Dell desktop.

5.2 Performance evaluation of INSERT

In this subsection, we report an empirical evaluation of INSERT. We compare INSERT with three strategies (T1, T2, and T3), which adopt a keyword-based approach, a URL-based approach, and a manual approach, respectively, as described below.

T1 (Keyword-based Recommender): One assumption of INSERT is that using pointers (URLs) together with the help of the pages indexed by a web search engine is more effective than using keywords alone to describe the target items. To validate this assumption, the first strategy (T1) we use for comparison is to provide keywords only as the input to a *full-text* search on all features of a content dataset. T1 uses the built-in function, match(), of MS SQL Server 2008 to implement the full-text search. It arranges the matched items in descending order of the matching scores returned by the SQL Server. To make a conservative comparison with INSERT, for each URL inputted to INSERT, we manually create a precise keyword, which is the actual and exact product name together with its model number that the URL aims to locate. Thus, the difference in the result between T1 and INSERT shows the effectiveness of INSERT.

T2 (URL-based Recommender): Apart from using the URLs as starting points, in Steps 3–4, INSERT also employs a corporate dataset to *refine* the items returned in the first round by a web search engine. To validate whether, and to what extent, this refinement step makes the recommendation list more effective, we

compare INSERT with a strategy (T2) which performs only Steps 1 and 2 of INSERT. Thus, the difference in the result between T2 and INSERT shows the effectiveness of Steps 3 and 4 of INSERT.

T3 (Manual Recommender): One may wonder what the time taken to locate the same product in a shop is if the user performs traditional retail shopping. We call the latter a manual approach (T3), in which either the customer or a salesperson serves as the (human) recommender.

Results of INSERT were obtained by executing our prototype system. The system retrieved the URLs kept in the last 7 days by the web browser, which was registered on an Android-based mobile phone and used Google as the web search engine. We implemented T1 as a separate program. To obtain results for T2, we disabled the implementation of Steps 3 and 4 of INSERT in our prototype system. For T3, an author (Chiu) of this paper acted as the customer to visit shops in a shopping mall in Hong Kong and to locate the products.

We then describe the dependent and control variables. Both *accuracy* and *time* are the dependent variables of the study. We define *accuracy* as the ratio of the number of correctly recommended items over the total number of items in the resulting recommendation list. For instance, if the recommendation list consists of 10 items, of which two are what we look for, then the accuracy is 2 ÷ 10 = 20%. The second dependent variable, *time*, is defined as the time taken for an implementation of a strategy to produce an output (recommendation list) starting from the acceptance of an input to the completion of displaying the recommendation list on the screen of the mobile client. To measure the time needed by a manual approach, we use a stopwatch.

There are several control variables in the study. The first control variable is the web search engine required by INSERT. We chose Google Hong Kong [21] because it was the most popular search engine

(a) (b) (c) (d) (e)

Figure 3. Proof-of-concept prototype (client side). From left to right:

(a) The application is initiated.

(b) After Step 4 of INSERT, the recommenation list is shown with its items grouped into categories.

(c) The user explores the DSLR category and the applcation shows two cameras.

(d) The user explores the first camera entries and the application displays the item details.

(e) After pressing the [Go] button on the details page, the application shows the zone where the customer may locate the item and examine it physically.

used in Hong Kong and all authors are located at Hong Kong. This search engine also served all Google search queries from the Mainland China at the time of performing this comparison. The second control variable is the threshold, α, in Step 2 of INSERT. We set $\alpha = 3$, based on our "shopping" experience on searching retail goods over the Internet in Hong Kong. The third control variable is the content dataset. We chose all the items on the notebook category pages of three large popular brands, namely, Fujitsu, Lenovo, and HP, in addition to the DSLR category of Sony. They were all extracted on 7 April 2010. The experiment used this combination because Sony also sold notebook computers, and HP sold digital cameras as well, whereas neither Lenovo nor Fujitsu sold any digital cameras. We collected all the products from the Hong Kong website of one major Hong Kong retail chain store. In total, there were 7 Fujitsu products, 10 HP products, 4 Lenovo products, and 6 Sony products in the content dataset. This content dataset was used by T1 to search for its recommendation items, and also by INSERT and T2. For the comparison between INSERT and T3, the dataset was further enhanced to include the products of Apple, Inc.

The experimental procedure of the experiment is as follows. To evaluate *accuracy*, we conducted six runs using different keywords (for T1) or URLs (for T2 and INSERT) to simulate a user's target items, which were: two HP computers, two Sony digital cameras, and two Fujitsu notebooks. All these models were in the content dataset described above, and we chose among the available models of each brand arbitrarily. For each run, we applied the input to the strategies and obtained the sequence of recommendation items. Since the keyword used by T1 was already the exact description of the item to be recommended, we used it as the target item. We then measured the size of the recommendation list and checked whether the target item was in the list. Based on these two values, we computed the accuracy of the recommendation list. We did not include T3 in this experiment due to a number of physical constraints. T3 required a human subject to visit a shop. In order to simulate the case realistically and control the factors, the same customer would have to visit the same shop six times to check for the goods. There would be learning effects by the customer on the shop environment and the salespersons might find the experiment unfair to them. Consequently, we did not include T3.

To evaluate *efficiency*, for INSERT, we measured the time taken to query a target item "Sony DSLR camera A850", and repeated this measurement for 50, 100, 200, 400, 800, and 2000 consecutive times in order to find out the variation of the time taken as the number of searches increased. Moreover, we cleaned up the URL history kept by the web browser before conducting each run. For T3, the customer visited two electronic shops in the shopping mall near his home, and located three items, namely, Sony DSLR A850, Lenovo ThinkPad T510, and Apple iTouch 32GB. Owing to the intrusive nature of T3, the number of runs was limited to two. We did not include T1 because T1 used a different tool and did not involve the use of a web search engine, which incurred network latency and delayed responses due to variations of web search engine loading and, hence, the comparison would not be meaningful. Similarly, we did not include T2 because T2 was known to skip Step 3 and Step 4. The time taken by T2 was known to be shorter than INSERT, even without doing an experiment.

The result on accuracy is shown graphically in Figure 4, where the *x*-axis shows the test case IDs in six trials, and the *y*-axis shows the accuracy of each trial. We observe that INSERT is better than both T1 and T2 in every case. Specifically, in each of four cases (namely, cases 1, 2, 3 and 5), INSERT was able to recommend only the exact item pointed by the input URL, giving an accuracy of 100%. In the other two cases (cases 4 and 6), INSERT recommended two items, one of which was the target item, giving an accuracy of 50%. On average, the accuracy of INSERT was 83.3%. For T1 and T2, they performed substantially less well than INSERT. In the last two cases (cases 5 and 6), they returned seven items in a recommendation list, only one of which was the target item, giving a mere accuracy of 14.3%. In the other four cases, the accuracy ranged from 16.7% to 33.3%. On average, the accuracy of both T1 and T2 was 19.2%. The big difference between the average values (83.3% for INSERT and 19.2% for both T1 and T2)

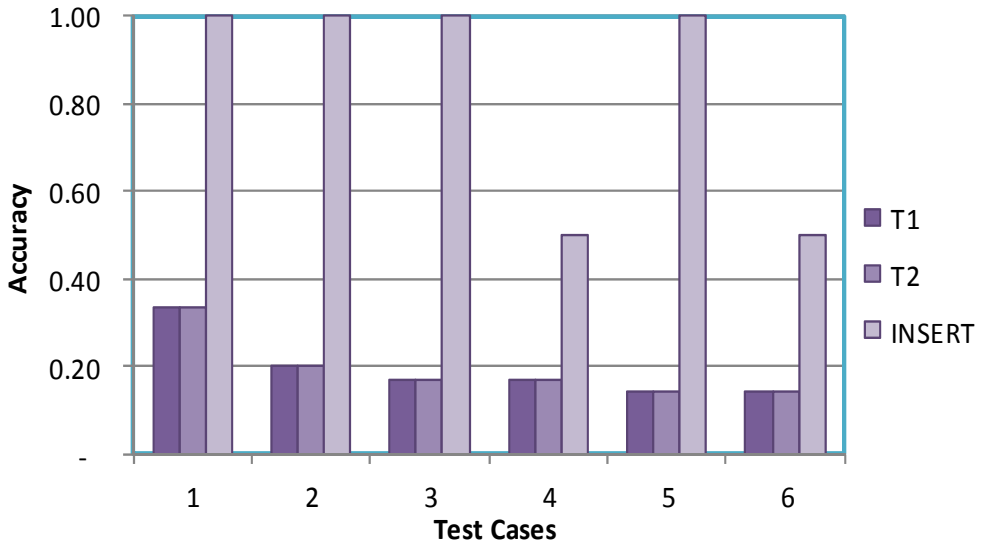

Figure 4. Accuracy of different strategies

shows that Steps 3 and 4 largely account for the improved accuracy. Furthermore, using the Mann-Whitney U statistical test to test the null hypotheses that INSERT is indistinguishable with either T1 and T2 in terms of accuracy, we found U = 0 and z = 2.8, showing that the null hypotheses can be rejected at a significance level of 0.01. In short, the performance difference between INSERT and T1 (and T2, respectively) is statistically significant.

The result on the time taken to complete recommendation lists by INSERT is shown in Figure 5. By simple calculation, the time taken to generate a recommendation list ranged from over 0.35 seconds to 3 seconds per query. The result indicates that it is feasible to obtain results from INSERT within a short enough period that a typical customer for ISS is willing to wait. Nevertheless, we noted that the time taken could be affected by many factors such as network traffic and the workload of the web search engine. None of these factors could be controlled by us. The timing results should be interpreted in light of these constraints and considerations.

For T3, we have interesting observations. While the author (who acted as a customer) searched for the item on a shelf, he was intercepted by a salesperson who enthusiastically offered help and described the products. He performed this trial twice, each instance at a different shop. One instance took him 961 seconds to locate the product and find its details in the shop successfully, and the other instance took him 377 seconds. The author also performed a search of the same products using our prototype, where the products were what he located by following the T3 approach. This time it took him only 254 seconds in total: 147 seconds to run the system and 107 seconds to read the specification of all the three items on the screen. This real-life experience indicates that INSERT has the potential to save much user time for in-store shopping.

There are a number of issues related to threats to validity. In the rest of this subsection, we discuss each of them. We used accuracy and time as the metrics in the experiment. Other metrics (such as precision and recall) to measure the effectiveness of a strategy are also feasible. Moreover, the time values reported in this evaluation section was subject to the variations in network traffic and the workload of the web search engine. None of these factors could be controlled or predicted by us. One may suggest using a static set of web pages and set up a search engine in a controlled environment to perform the measurement. However, doing so will defy the purpose of the experiment which aims to validate whether our strategy to delegate much contextual information about the users and user groups to real-life web search engines is viable. For

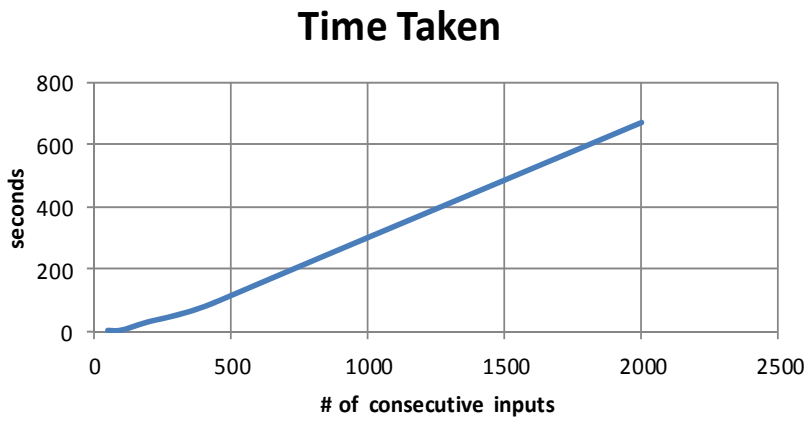

Figure 5. Time spent for INSERT

T3, it is a manual approach, and it locates different items owing to our environmental constraints. Therefore, we did not compare its accuracy against other strategies.

One may concern about the correctness of our implementation. While our integration of different algorithms and features as well as our treatment of data as pointers are novel, the algorithms themselves are not difficult to implement or verify. The most complicated algorithm for a recommender has been delegated to a web search engine, which improves the overall reliability of our implementation. In fact, this improvement in reliability is also a motivation of this paper to propose a service-based approach to recommender system. Another threat is the use of different kinds of inputs (e.g., keyword for T1 and URL for T2 and INSERT) in comparing the results from different techniques. In essence, the experiment used different starting points. It is hard to assess which one contains less meaningful information. On the other hand, T2 and INSERT accept the same inputs. The results show that their difference in accuracy is relatively large. Moreover, for T1, we have tried to use a conservative comparison by using the actual product name and model number that the URL is intended to locate.

To achieve broader generalization, a larger scale of experiment would be needed. In particular, the control variables have to be varied systematically. In the preparation of the reported experiment, we have tried our best to conduct it in a way as meaningful as possible, subject to the constraints of the present work. The time taken is also affected by how much optimization a tool has implemented, what programming language is used to implement the technique, and so on. A comprehensive design of controlled experiments that take into account the various controls and parameters is beyond the scope of this paper. Further work will need to address these issues.

5.3 Discussions

We have mainly demonstrated the applicability of INSERT in the context of in-store shopping, though we believe that the strategy is also generally useful in a variety of other contexts or business domains. For instance, it may be used by an English Learning Centre (ELC) to recommend materials for students who would like to improve their English. In such a scenario, based on the web sites recently browsed by the students via their smart phones, INSERT could instantly recommend books, magazines, newspapers, videos, or other reading materials kept in the ELC to students. Indeed, our work was also inspired by the use of Document Object Identifier (DOI) that universally identifies a scholarly document. Consider a device which has held a folder that contains a set of DOIs. Using INSERT, university libraries can retrieve the corresponding volumes and issues via an online web search engine, then match the combination of, say,

book title, volume and issue with the content database of the library, and finally suggest the location and call numbers of the required books. With the very large network of citations on these scholarly works and the journal information kept by the library's content database, many interesting applications that help users can be further developed.

Currently, we have only started to tap the potentials of INSERT. Admittedly, for deployment in practical systems for real-life use, some further enhancement and enrichment would be necessary. For example, partly to simplify the presentation of the core principles behind our work, this paper has not considered the privacy and security issues, which would be crucial for the success of real systems. Users of the client, for instance, should be reminded that their browsing history would be used by a backend server, and some mechanism needs to be in place to assure users that such information would be used for the agreed purpose only, and encrypted for preventing human access.

In addition, we have also made a number of simplifying assumptions so far. In what follows, we shall discuss how some of them can be addressed by extending the core strategy of INSERT.

First, INSERT starts by collecting the URL history of web pages visited by the user, which serves as the context for the context-aware recommender being implemented. Here it is assumed that the URL history has been kept by the web browser in the user's smart phone. While this assumption usually holds, there can be exceptions in practice. For instance, the page history can be non-existent if the user disables it in the browser setting. Even so, often the browser will still keep it for a short period, such as until the browsing session ends, and within this period INSERT still works. In fact, the precision of the recommendation can be even higher if the user has just finished doing web surfing on the target product items.

At the other extreme, the browser may have kept a huge list of visited pages, which will not only reduce the time efficiency of INSERT, but also its effectiveness in terms of precision, as many irrelevant pages are considered initially. Nevertheless, subsequent steps of INSERT will proceed to eliminate some ambiguous pointers, reduce the search scope and confirm the content with a corporate dataset. Therefore, we believe that the precision of the final recommendation list may still be better than simple keyword/URL searches. Our experiment has adopted the heuristics of extracting only the past 7 days' URL history as starting point, in anticipation that these recent visits are more relevant to the current search. It would be interesting to experiment with the effects of other options, such as a longer period or simply using the entire URL history despite its unusual length. Alternatively, this feature can also be enhanced by allowing a flexible period.

Indeed, the number of URLs available in the smart phone's web browser is in principle unlimited. In practice, this number is managed by the user or the web browser. It would be interesting to develop more advanced URL cleansing techniques than what we have proposed (Steps 3 and 4 of INSERT) in this paper.

INSERT has utilized the browser-stored URLs as pointer information because they can be extracted (once available) without explicit user actions. We note that, in principle, INSERT also works with the use of other pointer information. URL information is admittedly also less universal than location information as required by location-aware recommender systems (see Section 6 for some work related to these). Nonetheless, both URLs and locality information are subject to privacy issues. If no URL information is available, one may consider using other contextual information instead, but the resulting effectiveness is yet unknown. Further work and experiments may explore this dimension.

In describing INSERT, we have mainly used Google as an example web search engine to present how the returned index pages can be handled. As said before, other web search engines may be used as the underlying universal recommender. In some business contexts, domain-specific web search engines may be more appropriate. For instance, a search of research articles on Google Scholar may return a more focused set of pages than the same search on Google. We note that the structure of web pages returned by other web search engines may differ from what is shown in Figure 2. This is not a real issue, as long as some part of

the returned index page contains the information equivalent to what we refer to as index title in our illustrations in Section 4. Unlike database or information retrieval where standard query languages like SQL or markup languages like XML exist, there is no standard format of index pages returned by web search engines. In any case, it should be easy to adapt the implementation to these details as long as the information of the required main index entry exists. Furthermore, we describe INSERT based on the state-of-the-art user interfaces of web search engines. In the future, these interfaces may be modified and, hence, the essential information captured by Figure 2 may become invalid. Web search engines may be replaced by other technologies. Under these circumstances, some adaptive maintenance on INSERT may be necessary.

A URL string may contain fragments that are not valid for later uses on the same or other devices. For instance, it may contain a value standing for a session identity. A naïve web search with the URL string may not result in any hit. There are many approximate solutions, however. For instance, an adapted version of INSERT may modify a URL string by removing one or multiple parametric fragments (e.g., ?user=Mary, or ?sessionID=lkAsfahsfsa) at a time and then feed the modified URL string to web search engines. In principle, the combinations of parameter fragments that can be removed grow exponentially with the number of the parameter fragments. To deal with this problem, one may use meta-heuristics algorithms, which may find good solutions in a short time, to produce an adapted version of the URL string.

Another limitation of our strategy used in the retail situation is that some items may not be indexed by the web search engine in time. We have no good solution at this moment because it depends on the policy of a third party. In this case, the user may have to resort to the traditional shopping model (e.g., seeking help from the salespeople) to locate the items. However, our strategy may have helped the salespeople to handle the other situations already and, hence, hopefully reduce their workload.

Readers may also wonder how the URL strings can be sent to the server-side applications. For passive mobile device, it requires an installation of a client application (e.g., an iPhone application). Activation of the application, however, depends on the device setting, which is not within the scope of this paper. Ideally, a pervasive computing device will be able to discover and communicate with the server actively. It may require some semantic web support. It would be interesting to know how it can be done effectively.

Occasionally, INSERT may spend an indefinite amount of time waiting for a third-party service to locate and filter items. In practice, an implementation may incorporate a timeout mechanism and dynamically adapt the service composition accordingly (say, by switching to another third-party service).

6. Related Work

In this section, we discuss related work and compare it with ours. Generally, a recommender system [24][39] recommends information items to a user by filtering irrelevant or less relevant information items. Many such systems collect the number of times that a user visits a particular item as the basis of recommendation. For example, many academic journals provide lists of top-downloaded articles as recommendations to researchers. Some other systems employ more refined techniques, such as a *k*-nearest-neighborhood approach (e.g., [17]) to generate the top-ranked similar items and present them to the user.

There are three basic strategies for designing recommender systems. The first strategy is *collaborative filtering* or *social filtering* [38]. It collects the behavior of users on individual information items as the basis of recommendation, using techniques such as user-profiling [18]. Generally, *explicit profiling* requests users to provide personal information, such as age, gender and occupation, while *implicit profiling* monitors users' behavior without explicit user interaction [15]. The second strategy is *content-based filtering* [35], which bases its recommendation on the semantics of the information items (e.g., in ontology [1][49]) instead of users' behavior. INSERT is an instance of the third strategy, which is a hybrid of the first two [37]. Nonetheless, the above classification is based on the information usage. In this paper, we explore the differences from a software architectural viewpoint.

Our work has been motivated by the need for developing personalized recommender systems in a mobile environment, which is more challenging than that in the traditional domains [19]. For instance, Ge et al. [19] identified the challenge as due to "the complexity of spatial data and intrinsic spatio-temporal relationships, the unclear roles of context-aware information, and the increasing availability of environment sensing capabilities". To address these problems, the use of explicit ontology has been studied. For instance, *CHIP* [49] associates each art object with a piece of ontology, and employs users' rating on the art objects and topics in their content-based similarity matching. Blanco-Fernández et al. [8] recommended using both syntactic information on items and domain-specific ontology to improve the accuracy of the recommendation list. In contrast, INSERT requires no ontology at all.

Some researchers attempt to use unstructured data to infer more information for the implicit user profile. For instance, Lee et al. [29] pointed out the limitations of requiring an *explicit rating*. They used a data mining approach on the web log data to identify usage patterns as the basis for *implicit rating*. Similarly, Iwasaki et al. [25] used a Bayesian network to incrementally learn the usage patterns through historical data. In contrast, INSERT makes use of the URL traces to enhance the information based on the initial set of collected data. While CHIP [25] uses a lightweight and interactive user dialog approach to collect the initial user rating, INSERT is fully automatic once the initial set of URLs is available. Lee et al. [29] and Iwasaki et al. [25] assumed the presence of log data, whereas INSERT does not require this assumption. Yang et al. [54] proposed a machine-learning approach *PR*, which recommends a set of vendor web pages to PR users. The basic idea is to let each user browse web pages *through* PR, and based on the *content* of the visited web pages, PR recommends vendor web pages using similarity metrics that factor in the distance between the mobile user and the vendor location. To address the efficiency issue, PR keeps the location coordinates of all vendors by using an R+-tree in order to reduce the time taken to measure the distance between a user and a vendor. Unlike PR, INSERT does not mandate the user to access web pages via a particular browser or web infrastructure. Thus, our design of solution is more challenging than theirs. Another difference is that PR works at the individual web page level, whereas INSERT works with a set of URLs. Moreover, in our entire proposal, INSERT never needs to collect the *contents* of any web pages from the vendors.

The idea of domain-specific recommender system is not new. *Cyberguide* [2] is an early location-aware system that uses the current and past locations of mobile clients to recommend a set of services that fit the user's need. *Guide* [13] further elaborates on the challenges and experience in developing a context-aware system. In comparison, INSERT does not use location information as the starting point of recommendation. As illustrated in the motivating example, the use of location-based data does not help improve the precision of the recommendation list in the ISS scenario. Rather, INSERT uses the history of URL strings kept in the mobile client. Ricci and Nguyen [40] proposed to provide a feedback mechanism for users to inform a recommender so that it can refine the suggested list of items iteratively. The idea of using feedback to refine a basic ranking technique complements INSERT. Further work may explore whether and how INSERT can benefit from an extension to support a feedback mechanism by using a recursive service composition approach.

A recommender system can be used for multiple clients simultaneously. For instance, van der Heijden et al. [48] described a set of functional requirements of mobile recommender systems to support ad hoc discussions among a group of users, and Gupta et al. [22] proposed a method to identify ad hoc groups. Unlike these studies, INSERT has not addressed the problem of group-based visitors, each having a mobile client. It would be interesting to know whether the collaborative information available to the group of visitors may improve the effectiveness of the strategy. For instance, one way to extend INSERT to identify groups is to compute the similarity of the items being looked for based on their supplied URL strings, and then cluster the groups based on the similarity values and their social network relationships.

The data size of input profile can be an issue if the connectivity of the network is low. Miller et al. [34] reported their experience in developing a mobile recommender system whose clients may occasionally connect to a network. INSERT has not explored the multiple connection period problem, but only requires the sequence of URL strings kept in the browsers of the mobile client to be sent to the backend server. In practical scenarios, the amount of texts to be transmitted is typically small. For instance, even if there are 1000 URL strings, each containing up to 2048 characters, the amount of data transfer is merely 2 MB. By today's standard of WiFi and 3G bandwidth capabilities, the transfer may take a few seconds only.

Generally, a user model may vary as the type of device used, the characteristics of user preference, and the kind of interactions performed. *UbiquiTO* [12] was proposed to adapt the user model based on these variations. INSERT delegates this responsibility to web search engines to provide ranked lists of items. Lee et al. [30] reported that, in their experiment, incorporating the image item launch time and user buying time (and their combinations) for that particular image item in various ways help improve the accuracy of the recommendation list in 19 out of 24 cases. INSERT does not currently keep such dynamic information (i.e., the time of prior purchase transactions). To improve the accuracy of the recommendation list, INSERT uses a multi-phase matching approach to eliminate some false-positive items. The pipeline architecture [7] used in INSERT can be extended with an additional stage flexibly to incorporate such dynamic information.

Service composition approaches have been proposed before. Averjanova et al. [5] suggested combining a recommender with map services to enhance user satisfaction. Ge et al. [19] studied the idea of deploying recommendation systems in a sequence. Liu and Lee [32] proposed that some commercial web sites bearing the features of social network can be used to enhance the prediction accuracy of the recommendation list. Web search engines intrinsically use social networking features in their item ranking and, in this sense, INSERT has indirectly inherited this feature. Moreover, INSERT does not just use two recommenders in sequence but, rather, derives the recommendation capability of the higher level recommender from a universal recommender. In its second round of matching with the latter, INSERT further uses the associated keywords of individual items for matching. These keywords can be viewed as item-specific ontological information, and provide added values to the recommendation generated by the universal recommender.

None of these previous works has distinguished between *concrete data* (such as location, user preferences, or other contexts) and their potential use as *pointers* to a dataset. The difference is analogous to the difference between a SQL query and a dataset retrievable by the SQL query from a database. To the best of our knowledge, previous studies exclusively restrict their focus to the former, whereas INSERT recognizes the importance of the latter and applies it in its procedure to recommend items to users. While Chiu et al. [14] used the blog URLs to analyze user preferences, INSERT does not directly use the website information pointed by URLs.

7. Conclusion

A service-based application may discover and bind to such a recommender service to implement its own functionality. Nonetheless, owing to the cross-organizational privacy issue, the internal dataset of an organization is seldom exposed to external services. This issue poses a technical challenge on the use of an external recommender service that suggests choices based on an organization's internal dataset. To address this problem, we have proposed a strategy, INSERT, in this paper. The key idea is to have a higher level recommender that both accesses the internal dataset and invokes a lower level external recommender for item recommendation. Unlike the other approaches, the recommendation ability of the higher level recommender is *derived* from the lower level one. This paper has also interestingly proposed to use URLs as the user contexts. Our results show that deriving a higher level recommender service from an external recommender that overcomes the cross-organization privacy issue is feasible. Currently, we have only demonstrated the basic application of the key insights and practical ideas of INSERT, and it can be further

enhanced in a number of ways to address other practical issues, as discussed in Section 5.3. It would be interesting to further evaluate the extension of our proposal in these directions.

8. Acknowledgment

This work is supported in part by a Strategic Research Grant from City University of Hong Kong (project no. 7002464) and the General Research Fund from the Research Grants Council of Hong Kong (GRF project nos. 111410 and 123206).

9. References

[1] Aamodt, A., and Plaza, E. Case-based reasoning: Foundational issues, methodological variations and system approaches. *AI Communications*, 7 (1): 39–59, 1994.

[2] Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. Cyber-guide: A mobile context-aware tour guide. *Wireless Networks*, 3 (5): 421−433, 1997.

[3] Admvavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17 (6): 734–749, 2005.

[4] Alba, J., Lynch, J., Weitz, B., Janiszewski, C., Lutz, R., Sawyer, A., and Wood, S. Interactive home shopping: Consumer, retailer, and manufacturer incentives to participate in electronic marketplaces. *The Journal of Marketing*, 61 (3): 38–53, 1997.

[5] Averjanova, O., Ricci, F., and Nguyen, Q.N. Map-based interaction with a conversational mobile recommender system. In *Proceedings of the Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies* (UBICOMM 2008), pages 212−218, 2008.

[6] Baidu. http://www.baidu.com. Last access 11 Aug 2010.

[7] Bass, L., Clements, P., and Kazman, R. Software architecture in practice. 2nd edition. Addison Wesley, 2003.

[8] Blanco-Fernández, Y., Arias, J.J.P., Gil-Solla, A., Cabrer, M.R., Nores, M.L., Duque, J.G., Vilas, A.F., Redondo, R. Exploiting synergies between semantic reasoning and personalization strategies in intelligent recommender systems: A case study. *Journal of Systems and Software*, 81 (1): 2371–2385, 2008.

[9] Bhatnagar, A., Misra, S., and Rao, H.R. On risk, convenience, and Internet shopping behavior. *Communications of the ACM*, 43 (11): 98−105, 2000.

[10] Capra, L., Emmerich, W., and Mascolo, C. CARISMA: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29 (10): 929–944, 2003.

[11] CBSNews. Traditional gift shopping vs online. *The CBSNews*. Available at http://www.cbsnews.com/stories/2000/12/20/civilrights/main258563.shtml. Last access 26 Oct 2009.

[12] Cena, F., Console, L., Gena, C., Goy, A., Levi, G., Modeo, S., and Torre I. Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. *AI Communications*, 19 (4): 369−384, 2006.

[13] Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C. Developing a context-aware electronic tourist guide: Some issues and experiences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI 2000), pages 17−24, 2000.

[14] Chiu, P.H., Kao, G.Y.M., and Lo, C.C. Personalized blog content recommender system for mobile phone users. *International Journal of Human-Computer Studies*, 68 (8): 496–507, 2010.

[15] Chu, W., and Park, S. Personalized recommendation on dynamic content using predictive bilinear models. In *Proceedings of the 18th International Conference on World Wide Web* (WWW 2009), pages 691–700, 2009.

[16] CityNews. Online shopping vs in line shopping: Who wins? *The CityNews*. Avaliable at http://www.citytv.com/toronto/citynews/life/money/article/4144--online-shopping-vs-in-line-shopping-who-wins. Last access 30 Jul 2010.

[17] Cover, T.M., and Hart, P.E. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13 (1): 21–27, 1967.

[18] Gauch, S., Speratta, M., Chandranouli, A., and Micarelli, A. User profiles for personalized information access. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web — Methods and Strategies of Web Personalization*. Springer Berlin / Heidelberg, 2007.

[19] Ge, Y., Xong, H., Tuzhillin, A., Xiao, K., Gruteser, M., and Pazznai, M.J. An energy-efficient mobile recommender system. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD 2010), pages 899–908, 2010.

[20] Google. http://www.google.com. Last access 16 Aug 2010.

[21] Google Hong Kong. http://www.google.com.hk. Last access 10 Apr 2010.

[22] Gupta, A., Paul, S., Jones, Q., and Borcea, C. Automatic identification of informal social groups and places for geo-social recommendations. *International Journal of Mobile Network Design and Innovation*, 2 (3/4): 159−171, 2007.

[23] Happel, H.J., and Maalej, W. Potentials and challenges of recommendation systems for software development. In *Proceedings of International Workshop on Recommendation Systems for Software Engineering* (RSSE 2008), pages 11–15, 2008.

[24] Herlocker, J.L., Konstan, J.A., Terveen, L.G., and Riedl, J.T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22 (1): 5–53, 2004.

[25] Iwasaki, H., Mizuno, N., Hara, K., and Motomura, Y. User-adapted recommendation of content on mobile devices using Bayesian networks. *IEICE Transactions on Information and Systems*, E93-D (5): 1186–1196, 2010.

[26] B. Jiang, Chan, W.K., and Tse, T.H. Where to adapt dynamic service composition. In *Proceedings of the 18th International Conference on World Wide Web* (WWW 2009), pages 1123–1124, 2009.

[27] Jessica. *Disadvantage of Online Shopping.* Available at http://www.saching.com/Article/Disadvantages-of-Online-Shopping/3488. Last access 30 July 2010.

[28] Jones, J.A., Harrold, M.J., and Stasko, J. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering* (ICSE 2002), pages 467–477, 2002.

[29] Lee, S.K., Cho, Y.H., and Kim, S.H. Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. *Information Sciences*, 180 (1): 2142–2155, 2010.

[30] Lee, T.Q., Park, Y., and Park, Y.T. An empirical study on effectiveness of temporal information as implicit ratings. *Expert Systems with Applications*, 36 (2): 1315–1321, 2009.

[31] Liblit, B., Naik, M., Zheng, A.X., Aiken, A., and Jordan, M.I. Scalable statistical bug isolation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2005), pages 15–26, 2005.

[32] Liu, F., and Lee, H.J. Use of social network information to enhance collaborative filtering performance. *Expert Systems with Applications*, 37 (7): 4772–4778, 2010.

[33] Mei, J., Chan, W.K., Tse, T.H., and Merkel, R.G. XML-manipulating test case prioritization for XML-manipulating services. *Journal of Systems and Software*, to appear, http://dx.doi.org/10.1016/j.jss.2010.11.905.

[34] Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A., and Riedl, J. MovieLens unplugged: Experiences with an occasionally connected recommender system. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (IUI 2003), pages 263−266, 2003.

[35] Mladenic, D. Text-learning and related intelligent agents: A survey. *IEEE Intelligent Systems*, 14 (4): 44−54, 1999. DOI= http://dx.doi.org/10.1109/5254.784084.

[36] Park, J.-H. A recommender system for device sharing based on context-aware and personalization. *KSII Transactions on Internet and Information Systems*, 4 (2): 174−190, 2010.

[37] Pazzani, M. A framework for collaborative, content-based, and demographic filtering. *Artificial Intelligence Review*, 13 (5−6): 393−408, 1999.

[38] Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. Grouplens. An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 175−186, 1994.

[39] Resnick, P., and Varian, H.R. Recommender systems. *Communications of the ACM*, 40 (3): 56−58, 1997.

[40] Ricci, F., and Nguyen, Q.N. Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems*, 22 (3): 22−29, 2007.

[41] Robillar, M., Walker, R., and Zimmermann, T. Recommendation systems for software engineering. *IEEE Software*, 27 (4): 80−86, 2010.

[42] Roman, G.-C., McCann, P.J., and Plun, J.Y. Mobile UNITY: Reasoning and specification in mobile computing. *ACM Transactons on Software Engineering and Methodology*, 6 (3): 250−282, 1997.

[43] Salber, D., Dey, A.K., and Abowd, G.D. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems* (CHI 1999), pages 434−441, 1999.

[44] Tang, A., Babar, M.A., Gorton, I., and Han, J. A survey of architecture design rationale. *Journal of Systems and Software*, 79 (12): 1792−1804, 2006.

[45] Tasi, W.T. Service-oriented system engineering: a new paradigm. In *Proceedings of Service-Oriented System Engineering* (SOSE 2005), pages 3−6, 2005.

[46] Tse, T.H., Yau, S.S., Chan, W.K., Lu, H., and Chen, T.Y. Testing context-sensitive middleware-based software applications. In *Proceedings of the 28th International Computer Software and Applications Conference* (COMPSAC 2004), volume 1, pages 458−465, 2004.

[47] Unphon, H., and Dittrich, Y. Software architecture awareness in long-term software product evolution. *Journal of Systems and Software*, 83 (11): 2211−2226, 2010.

[48] van der Heijden, H., Kotsis, G., and Kronsteiner, R. Mobile recommendation systems for decision making "on the go". In *Proceedings of the International Conference on Mobile Business* (ICMB 2005), pages 137−143, 2005.

[49] Wang, Y., Stash, N., Aroyo, L., Gorgels, P., Rutledge, L., and Schreiber, G. Recommendations based on semantically enriched museum collections. *Journal of Web Semantics*, 6 (4): 283−290, 2008.

[50] Ward, M.R. Will online shopping compete more with traditional retailing or catalog shopping? *NetNomics*, 3 (2): 103−117, 2001.

[51] Wartena, C., Brussee, R., and Wibbels, M. Using tag co-occurrence for recommendation. In *Proceedings of the 9th International Conference on Intelligent Systems Design and Applications*, pages 273−278, 2009.

[52] Xu, C., Cheung, S.C., Chan, W.K., and Ye, C.Y. Partial constraint checking for context consistency in pervasive computing. *ACM Transactons on Software Engineering and Methodology*, 19 (3): Article No. 9, 2010.

[53] Yang, J. Web service componentization. *Communications of the ACM*, 46 (10): 35−40, 2003.

[54] Yang, W.S., Cheng, H.C., and Dia, J.B. A location-aware recommender system for mobile shopping enviornments. *Expert Systems and Applications*, 34 (1): 437–445, 2008.

[55] Yap G.-E., Tan, A.-H., and Pang, H.-H. Discovering and exploiting causal dependencies for robust mobile context-aware recommenders. *IEEE Transactions on Knowledge and Data Engineering*, 19 (7): 977–992, 2007.

[56] Ye, C., Cheung, S.C., Chan, W.K., and Xu, C. Atomicity analysis of service composition acoss organization. *IEEE Transactions on Software Engineering*, 53 (1): 2−28, 2009.

[57] Yu, T., Zhang, Y., and Lin, K.J. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Transactions on the Web,* 1 (1): Article 6, 2007.

[58] Zhang, Z., Chan, W.K., Tse, T.H., Jiang, B., and Wang, X. Capturing propagation of infected program states. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering* (ESEC/FSE 2009), pages 43–52, 2009.

[59] Zhang, Z., Jiang, B., Chan, W.K., Tse, T.H., and Wang, X. Fault localization through evaluation sequences. *Journal of Systems and Software*, 83 (2): 174–187, 2010.