RC24448 (W0711-262) November 30, 2007
Computer Science

IBM Research Report

User-Driven Mashups in Interactive Public Spaces

D. Soroker[1], Y. S. Paik[2], Y. S. Moon[2], S. McFaddin[1], C. Narayanaswami[1], H. K. Jang[2], D. Coffman[1], M. C. Lee[2], J. K. Lee[2], J. W. Park[2]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]IBM Ubiquitous Computing Laboratory
Seoul, Korea

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

User-Driven Mashups in Interactive Public Spaces

D. Soroker[1], Y.S. Paik[2], Y.S. Moon[2], S. McFaddin[1], C. Narayanaswami[1], H.K. Jang[2], D. Coffman[1], M.C. Lee[2], J.K Lee[2], J.W. Park[2]

[1]*IBM T.J. Watson Research Center, Hawthorne, NY*

[2]*IBM Ubiquitous Computing Laboratory, Seoul, Korea*

soroker@us.ibm.com

ABSTRACT

Searching and presenting rich data using mobile devices is hard given their inherent I/O limitations. One approach for alleviating these limitations is device symbiosis, whereby the interaction with one's personal mobile device is augmented by additionally engaging with more capable infrastructure devices, such as kiosks and displays. We have developed a software framework called Celadon, which builds upon device symbiosis for delivering zone-based services through mobile and infrastructure devices in public spaces such as shopping malls, train stations and theme parks.

An approach for rich data visualization that is gaining wide popularity is mashups. In this paper we describe User-Driven Mashups – a general methodology that combines device symbiosis and automated creation of mashups. We have applied this methodology to build a system that enables Celadon users to flexibly interact with rich zone information through their mobile devices, leveraging large public displays. Our system bridges public and personal devices, data and services.

1. Introduction

Consider the scenario where you are starting to feel hungry while in a shopping mall and want to find something to eat. Next to you is a large interactive display showing advertisements that don't concern you. You walk up to a nearby directory map and look under the "restaurants" category, but the name listing is not very meaningful, and for each name of interest you need to laboriously search for its location on the map.

Now consider the above scenario with the following differences. You use your personal mobile device to request to find restaurants corresponding to your preferences, whereupon these restaurants are highlighted on a mall map on the large display next to you. Through your mobile device you specify attributes of interest, such as cuisine type and price-range. Further interaction yields more details on selected restaurants, and also lets you book a reservation.

This paper describes a system for enabling the second scenario. Our system builds upon Celadon [23], which is a framework for delivering zone-based services through personal mobile devices and public infrastructure devices. The work presented here aims to support Celadon users in performing ad-hoc, loosely structured activities in public spaces. In such activities, the user has a goal in mind (such as finding a satisfactory restaurant) and needs to obtain specific information for making an informed decision. From the user's point of view, such a system must support intuitive and swift interactions, easily honing in on the information of interest. From an administrative point of view, such a system should permit easy deployment of many different informational services for seamless consumption by users.

A possible approach for supporting such a scenario is via text-based search such as used by the Google maps service for local business search. Such an approach is good for simple searches ("pizza"), but falls short for more involved searches ("inexpensive Italian restaurant"), since the user may not know the best search terms to use. Performing a sequence of refined text searches may be reasonable on the desktop but is far from perfect in a dynamic mobile scenario (due to the overhead of user interaction and network delays for each round trip). Thus we seek an alternative approach, in which the user knows what to specify.

Addressing is another complication: Google map locations correspond to street addresses, which are universally known and used, whereas zones such as shopping malls each have their unique addressing system, which may be unknown to parties unfamiliar with the zone details. Furthermore, street addresses are geocoded to a standardized 2D coordinate system (lat/long), which is used for associating data to screen location on a Google map, whereas lat/long is not a good solution for interior zones, since GPS may not work and a two-dimensional space for coordinates may not suffice. Thus we seek an alternative approach, in which the results can be displayed for arbitrary maps.

Mashups are becoming a prevalent paradigm for information composition in the Web. In particular, visual mashups show data superimposed in a meaningful visual context (such as in a map), and are thus very effective for conveying information (e.g., [8]). Typically mashups are explicitly programmed in advance, often by collecting data via a script and displaying it mashed up via APIs [7]. In the last couple of years, various tools (such as Yahoo Pipes [26] and QEDWiki [21]) have been developed for easy creation of mashups, also by non-programmers. Such tools create situational applications [25], which are easily constructed, lightweight, and serve specific purposes. Although these tools are designed to be easy to use, their purpose is to create applications that are subsequently run. In contrast, in the fast-paced scenario described here, the user's goal is to obtain information rather than create an application. Thus, to support these types of scenarios, mashups need to be formed on the fly, based on minimal input from the user. Such mashups are distinguished from situational applications in that they are not programmed by a human and are even more transient: they are quickly created, then used, then discarded. We call them *user-driven mashups*, and describe how they come to be in this paper.

The key contribution of this paper is a new approach for delivering mapped zone information. Our system employs the concept of user-driven mashups and addresses the following challenges:

- End-users can easily specify the information they wish to map, even through their mobile devices
- Providers can systematically deploy data services for supplying information and zone maps for mapping the information
- The system automatically associates data with its location and visualizes it in on arbitrary zone maps without programmer intervention.

According to our approach, data service queries are specified via a GUI on the mobile device, and the search results are mashed onto a zone map and presented on a large display. Beyond our current implementation, further enhancements support the use of preference information for inferring data of interest, as well as distributed and collective mashups, wherein multiple devices and users are involved in specifying the mashup.

It is important to explain how our work is distinguished from existing Web sites, such as Trulia [26], which let the user easily specify the desired features of the information to be mashed up (in the case of Trulia, the data service provides listings of homes for sales and the GUI lets you select relevant parameters such as neighborhood, price range, size, etc.). The distinction is that such Web sites are explicitly programmed with a specific GUI for a specific data service mashed up in a specific way, whereas in our system both the GUI and the mashup logic are constructed dynamically at runtime for a wide range of different data services.

In this paper we describe the approach, architecture and design of the system, as well as pertinent details of the implementation. We describe our experience to date and then provide a deeper discussion of some issues, including comparison with other approaches, such as text-based search, and outline future directions. We close the paper with a discussion of related work and conclude.

2. Approach & Architecture

This section describes our approach for the User-Driven Mashups system (UDM) and its architecture. Since UDM builds upon Celadon, we start with a simplified version of the Celadon architecture that best helps explain the structure of UDM.

2.1 Celadon Architecture

UDM is built on top of the Celadon distributed architecture, whose physical computing nodes consist of *mobile devices (MD)*, *facility devices (FD)*, and a *zone collaboration server (ZCS)*. MDs are carried by Celadon users, and get associated to the zone upon entry. FDs are fixed devices associated with the zone, such as large displays and kiosks. The ZCS is the "central brain" of a Celadon zone, responsible for managing the zone data, including its associated members (MDs and FDs) and services.

RESTful data services are a central building block in Celadon's software architecture. Data is retrieved from such a service by passing it an appropriate query via an API call or an HTTP request. The ZCS maintains core zone information, on members, services, location and interaction, via several core data services. Thus, for example, to get a listing of all services currently registered in the zone, clients send a wildcard query to the service-info service hosted on the ZCS.

2.2 UDM Process and Architecture

The central operation in UDM is to obtain zone data of interest and present it superimposed on a zone map. The basic process for achieving this is as follows: the MD presents to the user relevant data services, zone maps and FDs that can be used. Once the user selects the elements desired for the mashup, an interface is presented on the MD for specifying mashup details. Upon submission, the mashup is computed, and the results are displayed on the selected FD. The user can then refine the mashed

data and interact with the displayed mashup results via the mobile device.

The UDM architecture is shown in Figure 1, in which rounded rectangles are components specific to UDM. On the MD, the *service explorer* lists relevant services, and the *mash detailer* presents a GUI for specifying the details of the data to be mashed up. On the FD, the *mash renderer* presents the data of interest superimposed on a zone map. The *mash maker* (which may run anywhere and therefore is drawn outside any dotted rectangle) computes the mash results for presentation on the renderer.

2.3 Data services and zone maps

Celadon data services have a regular structure employing a RESTful design [14], and this regularity is leveraged in our UDM methodology. Each data service provides its data as a list of records, all of which adhere to a consistent record schema. Data services are self-describing in that they also provide the schema for their records as well as for queries against those records. A given data service schema may support multiple record types. As a convention in our current implementation, a record schema contains zero or more *fixed fields* and zero or more *attribute assertions*. Fixed fields are typically used for singleton record values, such as name or ID. Attribute assertions are more free-form, and can be used in cases where a record may contain multiple values, such as phone number (when an establishment has several phone numbers). Each attribute assertion is a <subject,verb,object> triple – where the subject contains an attribute name, the object a value, and the verb is a comparator. For example, <cuisine.“=”,“Korean”> denotes a Korean cuisine (for a restaurant info data record), and (floor,“>”,5) means a location above the 5th floor.

An important characteristic of our system is that we use the same notation for both data records and data queries. In other words, a query is represented as a data record, with fixed fields and attribute assertions that are matched against the set of records provided by the data service. Thus, for example, if the attribute assertion (cuisine,“=”,“Korean”) is part of the data query, then, by a process of matching, the data records of restaurants whose cuisine is Korean will be obtained when submitting this data query. When a query contains multiple assertions, they are combined via logical operators.

Celadon data services also accommodate listeners, which are notified upon changes to the data (addition, deletion or modification of records). This feature is used in several places in the UDM subsystem, such as when there are changes to data the user has selected for mashup.

Zone maps contain a structured graphic and one or more directories. In the graphic, regions of interest (such as regions corresponding to stores in a mall map) are associated with unique physical location IDs. Each directory associates various metadata with the physical locations. Such metadata may be logical

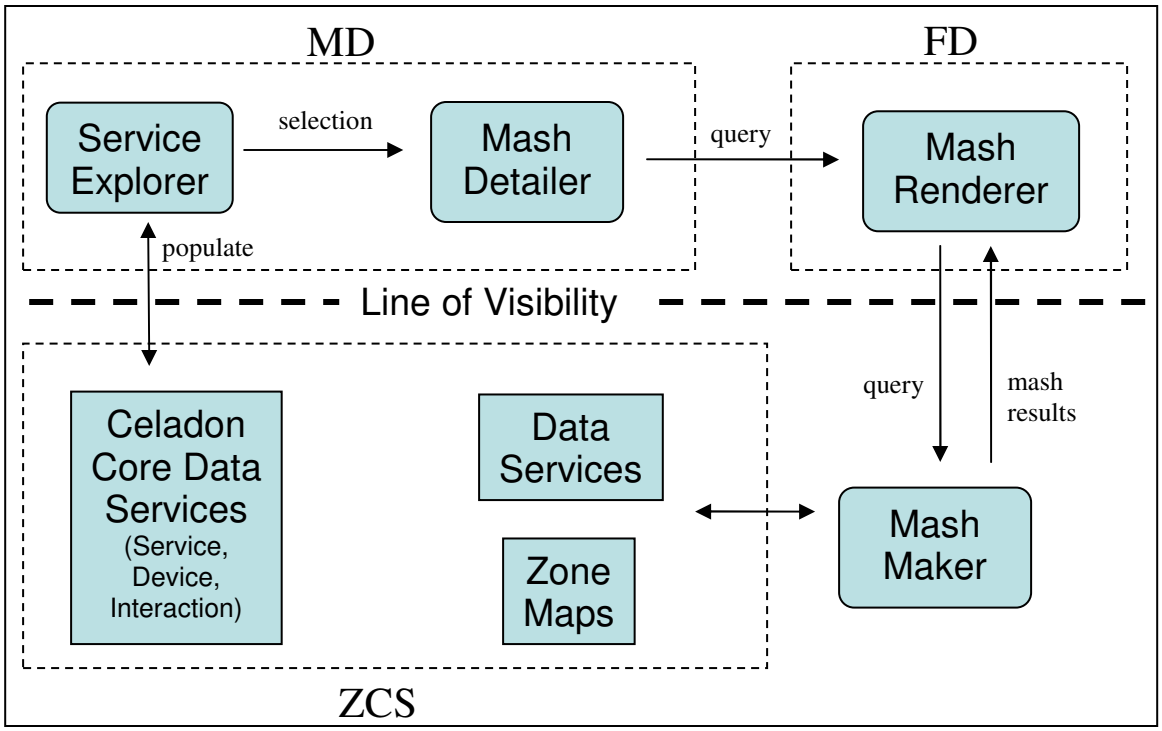

Figure 1: UDM Architecture. Components above the line of visibility are seen by the user.

location information or other information associated with that physical location such as a store's name or phone extensions. We assume that zone maps are maintained by the zone administrator, so that when there are changes in layout or contents of the zone, the graphical map and associated directories are updated accordingly.

2.4 Service Explorer

The starting point for the UDM process is to expose available services and resources to the user. This is done through the service explorer, running on the user's MD. The service explorer gets its data by querying core data services on the ZCS to retrieve listings of devices, zone maps, and data services. A listing of large display FDs and their status is obtained by querying the member-info service and interaction-info service, whereas listings of zone maps and data services are obtained from the service-info service. To reduce resource consumption, these queries may be tailored to the user's preferences or context (such as location), for example retrieving only displays in the user's general proximity, or data services in line with the user's interests. Further tailoring is done at the presentation layer, where the user can sort resources in various ways (relevance, availability, name, category, popularity, etc.).

To advance to the next step of the UDM process, the user needs to select a facility display, zone map and data service for mashup. When selecting a zone map, the system limits the choice of data services to ones compatible with the selected zone map. For example, services containing information about the entire shopping mall may differ from ones for a department store within the mall.

2.5 Mash Detailer

Specifying the information of interest on the mobile device is done via the mash detailer. Its design relies on the regular structure of Celadon data services, whereby it presents an interface for creating queries for the selected data service. The mash detailer fetches the query schema and dynamically constructs a GUI based upon it. The GUI exposes both fixed fields and attribute assertions, providing as much structure as is available in the schema. For example, a restaurant info query may present drop-down lists populated with known cuisine types and price levels (Figure 4). An additional role of the mash detailer is to let the user control how the mashed data is displayed on the zone map. For example, information may be displayed by highlighting regions of interest or, alternatively, via markers.

An alternative approach to asking the user for the mashup details is to infer them from the user's preferences available on the MD. This approach, as yet unimplemented, is expanded upon in Section 5.5.

2.6 Mash Maker

The mash maker is responsible for creating and managing the *working set*, which is the list of mashed-up records. Each element of the working set is a data record augmented with its physical location within in the zone map. The main process of the mash maker is as follows: it obtains the query created by the mash detailer and sends it to the selected data service; it then takes the set of records returned by the data service and correlates them with the zone-map directory info, so as to obtain the physical location of each data record, thereby constructing the working set; finally it sends the working set to the mash renderer for display on the FD.

Stepping back we see that the fundamental operation of the mash maker is the composition of two XML record sets, which we frame as a two-stage pipeline: a join followed by a transform, as shown in Figure 2.

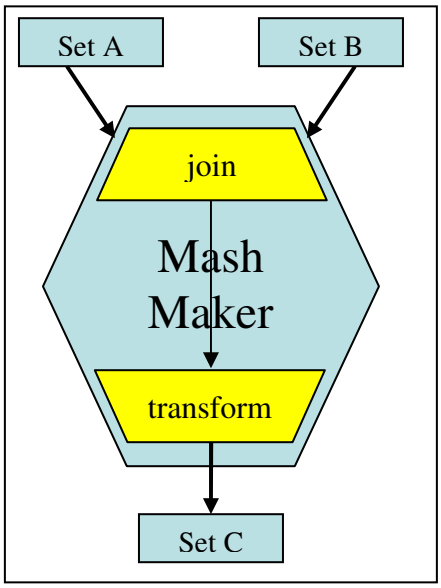

Figure 2: Mash maker design

In the join step, for each element of set A, a matching element of set B is found, and the information of the two matching records is joined. A general mechanism for modeling matching rules is XPath expressions [31], where such an expression would specify what constitutes a match (e.g., the value of field F_1 in a record from set A should be equal to the value of field F_2 in a record from set B). In the transform step, joined record are transformed to a given target format, which can be modeled with XSLT templates [32].

Architecturally, the mash maker core is an abstract component, which becomes concrete by plugging in specific modules for the joiner and transformer. Two such modules are XPath and XSLT processors as indicated above. In our current implementation we have chosen more limited modules, for reasons discussed in Section 5.4.

2.7 Mash Renderer

The mash renderer displays the working set on the FD, superimposed on the zone map. It receives the query object from the mash detailer, which contains specifications of the data service, the zone map, and details the data of interest and how to display it. It displays the zone map, relays the data query to the mash maker, and superimposes the working set returned from the mash maker. It then communicates with the MD to receive further instructions, which could be graphical (zoom / pan), contextual (fetch menu related to selected working-set object), new or refined query, or session termination.

Additionally, the mash renderer adds a listener to the selected data service so that it is notified of any underlying data changes, in which case it re-invokes the mash maker to obtain an updated working set and refreshes the display.

2.8 The UDM Ecosystem

UDM involves several parties: zone administrators, providers of information services, providers of physical services, and users. This section describes their roles in supporting a successful UDM deployment. For concreteness we will use a shopping mall as an illustrative example of a zone

Zone administrators manage a Celadon deployment and are responsible for zone-wide information. They provide and maintain zone maps for zone-wide information (e.g. maps of entire floors of the mall), which includes their graphical representation and associated directories. When a change in the zone occurs (e.g., changing ownership or contact information of a store in the mall), the zone administrator should update the relevant zone maps accordingly. Zone administrators also host the core data services of the zone, including one listing available services: any data service or zone map need to be registered with this service-info data service.

Providers of information services create and maintain data services, whose information content need not be limited to the zone. For example, a restaurant information service may provide information about many restaurants, both in the mall and outside it. They are responsible for keeping the information accurate and up-to-date, and for registering the data services they provide with the zone.

Providers of physical services (store owners in the mall) need to ensure that they are represented fully and correctly in zone maps and data services. They may have a relationship with providers of information services so as to ensure that their data is kept up to date. They may also contribute their own data services and zone maps to the zone. For example, a department store within the mall may provide zone maps of its own floor plans.

Users need to have the Celadon client code installed on their mobile device, and will likely need to have a Celadon account (which may be free of charge). In order to fully enjoy Celadon's benefit, they need to become familiar with its various offerings such as UDM.

3. Implementation

In this section we discuss some aspects of our current implementation of the UDM architecture.

Celadon clients (MDs and FDs) are built on the Lotus Expeditor platform[12], which extends the Eclipse Rich Client Platform (RCP). The Expeditor runtime is based on OSGi [19], which is a container for Java components called bundles. On the FD we leverage the embedded browser feature of RCP for a browser-based implementation of the mash renderer that is primarily implemented in JavaScript.

Service Explorer: Figure 3 shows a service explorer screen shot, listing a facility display, a service, and a user's mobile device. As such it provides a view of available resources in the zone. Celadon's dynamic service discovery feature is leveraged for populating the service explorer. Here the mashup viewer service is shown as a service available through the facility device. After the user selects a service in the service explorer panel, the MD fetches the appropriate application bundle from the server if not available on the device (using OSGi dynamic loading). Upon selection of the mashup service, a local controller is launched on the MD, which runs the UDM process as described in Section 2.2.

Figure 3: Service explorer screen shot

Mash detailer: Figure 4 shows a mash detailer GUI that was automatically constructed for our demo restaurant info service.

Figure 4: Mash detailer screen shot

The fixed fields are: Location ID, Operating Hours, and Name. Note that the data service record may have additional fields that the service designer may choose not to expose in its query schema (such as low-level key fields). The attribute assertions are located in drop-down combo lists below the fixed fields. Upon selection of a subject (e.g., "Cuisine" in this example), the verb and object fields are populated. By clicking the small arrow button, the user adds an assertion.

The attribute assertions are collected and displayed in a text area, where they can be edited manually by the user if needed. Assertions for different attributes are put in separate text lines, whereas assertions for the same attribute are concatenated on the same line, to denote a logical OR. In the example shown in Figure 4, the user wants to see moderately priced restaurants whose cuisine is either Korean or Chinese.

Mash maker: we have implemented an extensible mash maker (according to the design of Figure 2), into which different joiners and transformers can be plugged in.

Rather than employing a full-fledged XPath processor, our current joiner implementation uses *match-lists*, where a mach-list is a list of field names, and two records match if they agree on a match-list, i.e., the values of each of their named fields agree. For example, for the match-list {"name"}, the two records must agree on the value of their "name" field, and for the match-list {"zone", "cell"} the records must agree on both their zone and cell values. Match lists are associated with zone map directories.

Our current transformer implementation is, as well, more limited and lightweight that an XSLT processor. It uses heuristic rules to rearrange data from the joined records into a format that is expected by the mash renderer.

Mash renderer and zone maps: Our mash renderer is browser-based, implemented as a collection of JavaScript components for the core display functionality, wrapped in a Java RCP container for communication with the MD and ZCS. Its code structure is shown in Figure 5.

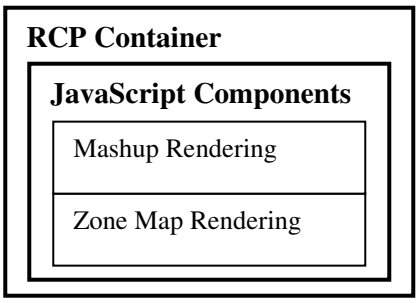

Figure 5: Mash renderer code structure

We use Scalable Vector Graphics (SVG) for the zone maps, where regions of interest in the SVG are annotated with unique identifiers that serve as physical location IDs. Each element of the working set contains a physical location ID, and it is used by the mash renderer to determine where that element should be superimposed on the zone map.

The zone map rendering code, as well as mashup rendering is also used by other parts of the Celadon system, primarily as part of zone administration, where zone managers can monitor zone status and activity. This code also contains enhancements for helping in visual construction of zone maps and associated directories.

Sample computed mashup results, as displayed by the mash renderer, are shown in Figure 7 and in Figure 8. In these displays the mashed information is shown as informational markers on the zone map.

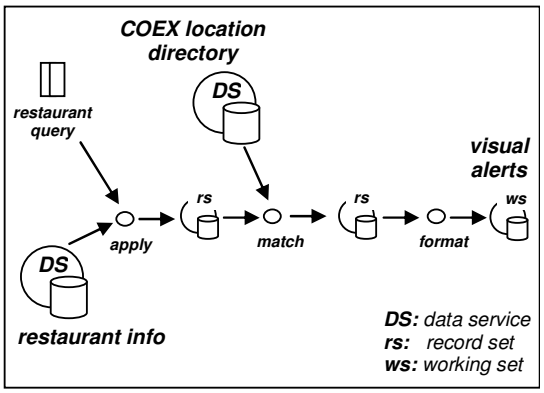

Figure 6: Data flow for mall mashup

Performance considerations: to help achieve good performance, especially on the MD, we have used lightweight components. For efficient messaging we use MQTT [15], which is optimized for resource-contrained devices. We employ a simple HTTP client for accessing Celadon data services rather than a full web services stack. For schema processing, which is needed by the mash detailer, we have written a limited pull-parser based schema processor, which is an order of magnitude faster than the EMF-based general-purpose schema processor. Our implementation of the mash maker is also lightweight, and can be run on the MD to support other scenarios.

4. Evaluation

We built the UDM system as described in the previous sections, and evaluated its behavior with concrete devices, data services and zone maps. This section describes our two main evaluations of UDM, including their setup, results and lessons learned. The first evaluation was in a lab demonstration setting, and the second was in a public demonstration setting; evaluating UDM with users and service developers is an important part of our future work.

4.1 Evaluation Setups

4.1.1 Shopping mall demonstration

In a lab setup we constructed a zone map for a huge upscale fashionable underground shopping mall attached the COEX (Convention and Exhibition Center) in Seoul, South Korea (as shown in Figure 7). The mall covers 7865sq meters and includes electronics, books, media, retail, department, clothing, and toy stores, an aquarium, a movieplex, and a host of restaurants. This mall is very busy during evenings and weekends and most visitors have at least one mobile device with them. Visitors may have trouble navigating the mall because of its size and complexity. In addition, many signs are in Korean and requests for navigational help from foreign visitors to the mall often end with no useful result

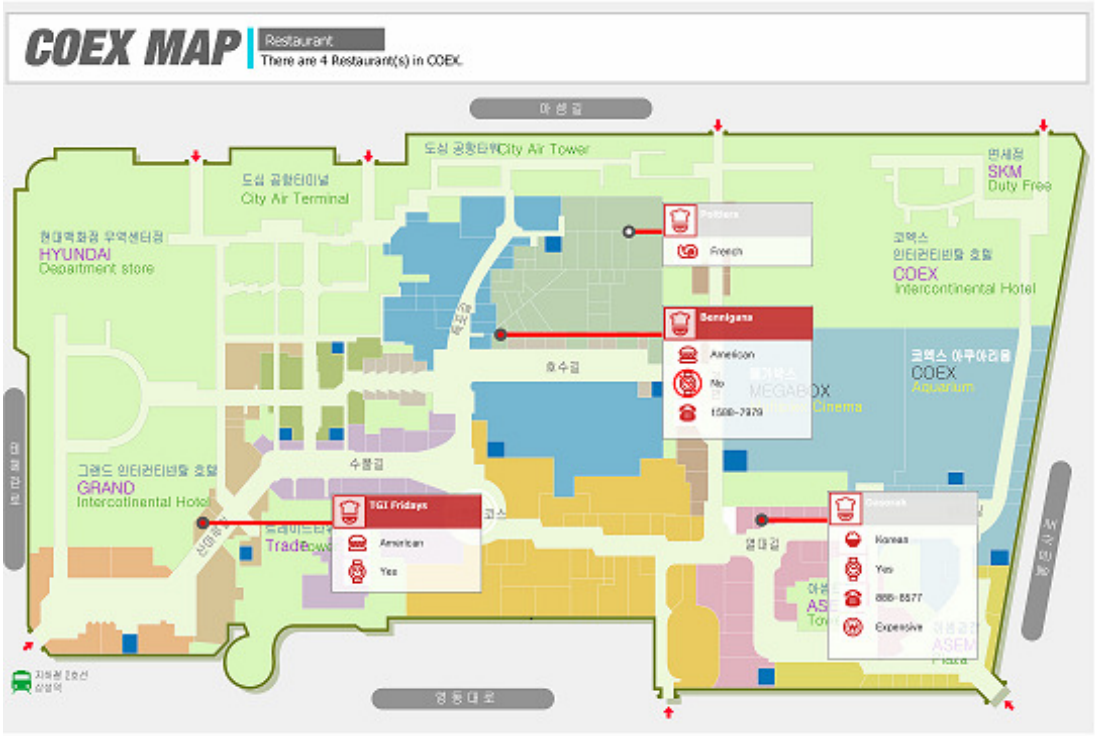

Figure 7: Restaurant-information mashup in the COEX mall.

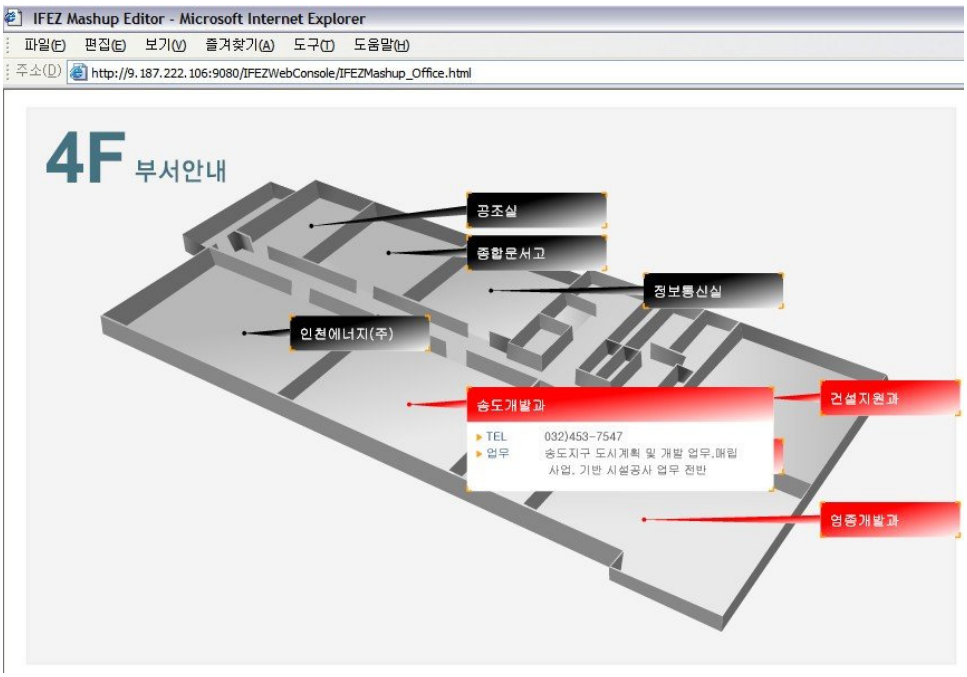

Figure 8: Floor-plan mashup in the IFEZ public demonstration

due to language issues. An informal survey of local visitors revealed that though the issues they faced were fewer, they were convinced that they could benefit from user driven mashups to present relevant information on large displays. Towards this goal, we started with two data services, which provide information on restaurants and stores in the mall. We then learned that handicapped individuals had trouble finding public restrooms in the mall that they could use. So we added a restroom locating service as well. The restaurant service allows users to specify parameters such as cuisine and price range. The store service lets specify the type of store (such as book store, sports goods, etc), and some additional details about the type of goods they are interested in.

Figure 3 shows a mash detailer screen for the restaurant-info service, and Figure 6 explains the data flow involved in mashing up restaurant data on the COEX map: a query is applied to the restaurant-info data service, which results in a record-set that is matched against records of the COEX location directory data service; the resultant record set is formatted to produce a working set in the form of visual alerts that are displayed as shown in Figure 7.

4.1.2 Ubiquitous City Demonstration

Our COEX lab demonstration helped win an engagement with the Incheon Free Economic Zone (IFEZ), which refers to the areas designed as centers for international business within Incheon, a city of thriving business and commerce in Korea. Facility displays for several zone maps we constructed are currently installed in the central government office building – in the lobby as well as on several floors – with an eye towards subsequent rollout on a metropolitan scale.

We have deployed two different MD types in IFEZ: Windows Mobile PDAs (HP iPAQ) running the Java RCP platform described above, and smart phones (Samsung SPH.M4500) for which we used native C++ code for component implementation. Both the PDAs and smart phones use MQTT and XML for communication over an 802.11 wireless channel. They both use the same protocols to interact with facility devices and external UDM components.

Each facility device comprises a mini PC (AOPEN DE-945FX) attached to a 60-inch LCD display. All FD applications are Java-based bundles running on the desktop edition of IBM Lotus Expeditor (which is an enhancement of Eclipse RCP).

We implemented three IFEZ zone maps, for showing attractions, a building guide and organization chart. Attraction results show corporate entities superposed on a map of the different IFEZ districts (such as Songdo, Cheongna and Yeongjong). For the building guide service, floor plans serve as the zone map and a building directory data service provides information such as which stations are populated and the kind of tasks performed at the various stations. An example of a floor plan mashup is shown in Figure 8. The organization service uses an organization chart as the zone map, and displays superposed data on the personnel such as contact information and job description.

In this trial deployment, MDs are loaned from the help desk at the IFEZ government building (rather than owned by users). After the MD connects to the Celadon environment, the visitor may select a service on the MD. Upon selection of the mashup service, a choice of the three aforementioned zone maps is provided. Upon selection of a zone map, it is shown on the associated FD display, and mashed up data is computed and displayed by following the UDM process described in Section 2.2. Further interaction with the displayed mashed data can be done through the MD or directly on the FD.

4.2 Evaluation Results & Lessons

We assessed UDM for usability, performance, ease of developing artifacts, and ease of administration.

4.2.1 Usability

A couple of potentially confusing aspects became apparent in the public demonstration. The first was the role of the mobile device: it is natural for a small form-factor device to serve as a remote control for a large display. However, in UDM the MD also provides its own rich display functionality thus competing for user attention. The second unclear aspect was the way a facility device is chosen – via selection on the mobile device – as opposed to direct interaction such as pointing to the large display with the MD or RFID-type interaction such as touching or swiping. Because of these reasons, some users did not easily accomplish their intended tasks, though training did not take much time or effort.

We also got some feedback from IFEZ users, that the interaction with the displayed mashup is not sufficiently refined and dynamic. Specifically, it was suggested that when selecting a certain point in the working set, magnification of that portion of the map or other dynamic visual response could enhance the experience. Some suggestions were to use Flash animation or audio/video. In fact, incorporation of multimedia files is supported on the SVG 1.2 spec. We use the Adobe SVG viewer plugin, which supports proprietary tags for Flash. Thus, our system can be evolved to incorporate richer multimedia into the mashup, which may improve user satisfaction.

4.2.2 Performance

In preparation for deployment, we first focused on identifying performance bottlenecks in the individual components of the UDM pipeline. Response times of the mash maker and mash renderer were instantaneous in our trials, and the only bottleneck identified was on the MD, when launching the mash detailer. In some cases, initialization time exceeded 10 seconds on an iPAQ hp2790b running Windows Mobile 5.0. We identified the cause of the problem to be the schema processing module, based on the Eclipse Modeling Framework (EMF). We implemented a less general but significantly lighter weight module for schema processing based on the KXML pull parser, and subsequently got the mash detailer initialization time to be consistently less than a second.

4.2.3 Developing artifacts

For developing the artifacts we used a combination of existing tools and tools that we developed as part of the Celadon project.

In our implementation, mashups are visualized via SVG data sets, and are geared towards structured representations of indoor spaces such as stores and shopping malls. However, our approach of applying data transformations also applies to other mashup usages such as geographically encoded data sets and organizational data. The SVG approach for zone maps fits well with the concept of store planograms [20], which capture the key elements of a store layout as individually identified graphical elements. Planograms are widely used in the retail space to manage and plan the placements of product sets at key locations.

Production of this style of mashup involves (1) the importation of zone map data from preexisting sources, (2) the identification of key zone map elements as output targets for the mashup, and (3) the construction of directories that map from a domain oriented data space to the identifier space of the zone map. A full zone map can be very detailed and involve thousands of graphical elements. In practice, however, only a few dozen are needed for an adequate mashup experience. Thus, step (2) is a key step in reducing the overall complexity of creating the artifacts and of computing mashups for them. The tedium of this step can be reduced by preparatory editors (we have under development) which allow the user to browse the imported diagram using the mouse, selecting only key elements to be used as mashup targets. In some cases, an image is imported as the underlying representation. In this case, invisible bounding polygons are constructed by a preparatory editor and used as the mashup targets. Step (3) involves the construction of data elements which map domain identifiers (e.g. phone numbers, business names, or logical location identifiers such as "store7/aisle3/shelf4/bin7") to renderable identifiers (e.g. "shape72"). These mapping elements are aggregated into location directories, which act as data

sources for intermediate transformations in the mashup process.

The three informational data services were developed in part using the Celadon data tools, which take as input a service record schema and generate various code artifacts, including a server-side JDBC service implementation, and specialized JavaScript and J2ME clients. For our purposes the main use of the data tools was for the server-side implementation and JavaScript client, since the UDM components use a generic J2ME client for querying data services (in the mash detailer and mash maker).

Since we observed that there is an overhead for each data service implemented (e.g., using the data tools, maintaining RDB tables), we looked for a simplifying approach. Our solution was to construct the three data services as a single "shopping info" data service, with three different query types in its schema (for restaurants, stores and toilets, respectively). We further simplified by having a single record type, in which the fixed fields are common to all three services, and the attribute assertions are service-specific. The schemas of the shopping information record and the restaurant query are shown in Figure 9. Note that the query schema only exposes relevant fixed fields (shop name, operating hours and location), and hides other fields (shop ID, which is not externally known, and shop type, which is fixed as "restaurant" for restaurants).

Repackaging several services into one helped reduce administrative overhead, but did not diminish the validity of demonstrating and testing UDM for multiple services, since each of the three services was listed as a separate service (with all listings pointing to the same URI). Our ability to do so without having to affect the architecture highlights the flexibility of the RESTful design of our data services. This repackaging is generalized to a technique that has been added to our arsenal of best practices for future deployments.

4.2.4 Administration

To help in administering data services, the Celadon data tools also generate JavaScript clients that provide a browser-based interface for querying, adding, removing and modifying service records. The specification panel of such a JavaScript panel

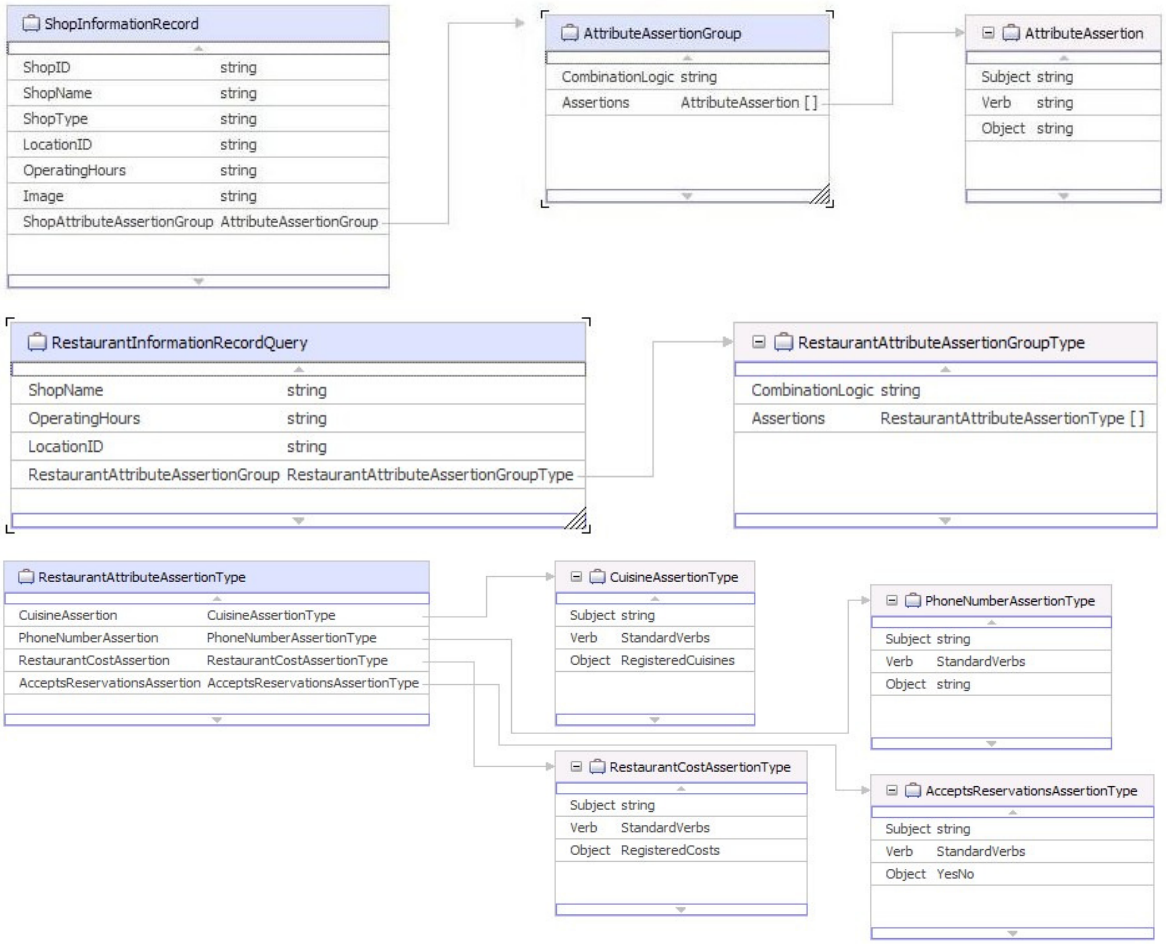

Figure 9: Schema for shop record and restaurant query within the shopping data service

has a similar design to that of the mash detailer, with additional controls for specifying the desired function (the mash detailer's single function is "query"). In the current implementation the generated JavaScript clients are service-specific. An alternative approach we are considering is to have a single generic client that, similarly to the mash detailer, inspects the service schema at runtime to produce an interface that is tailored to that data service.

5. Discussion & Future Work

In this section we discuss several topics of interest related to our UDM work.

5.1 Addressing within zones

When working with geographical maps there is a universal coordinate system – latitude and longitude – which is used to tie data to location, possibly through intermediate services (such as conversion of street addresses to lat-long). One of the challenges in UDM is the lack of such a universal addressing scheme for arbitrary zones. We solve this by introducing an intermediary layer in the form of directories that associate logical to physical locations. Since the physical locations (SVG region IDs) are too low-level to be known by external data services, they need to be correlated with higher-level information, and that is the role of directories. A novel aspect of this design is that *any* information can be associated with the physical locations. Since a directory refers to physical locations, it needs to have intimate knowledge of the zone map, and thus needs to be created and maintained by the owner of the zone map, which is typically the zone administrator.

A zone map contains the x-y coordinates of each region in its own coordinate space (as an inherent part of the SVG data). This information can be employed for proximity computations, such as sorting working sets according to their distance from the user.

5.2 Public vs. private information

The UDM process described in this paper involves both the user's personal mobile device and large shared displays. As such, the process is part private and part public, and it is important to keep sensitive information private. In our design, the publicly displayed information is a mashup of publicly-available information from data services and zone maps. Even though the data is public, the *intent* of the user may be surmised by others viewing the mashed result; the extent to which users' intent is sensitive for such mashups should be studied in actual deployments. In more elaborate scenarios, where the user query is constructed based on user preferences or other information stored on the MD (such as a shopping list), more care may need to be taken in keeping private information private, and techniques such as those proposed in [3] could be employed.

A current feature of our design is that user input is done though the personal mobile device, which is advantageous from a privacy perspective, but may be overly restrictive. In certain configurations it may be natural to support direct interaction with a touch-sensitive display for zoom, pan, and selection. When a mash point is selected, more detailed information may be presented on the large display, or pertinent data may be sent back to the user's mobile device.

5.3 Shared resources

The UDM process may create contention when multiple users want to use a large display simultaneously. How this would affect adoption is hard to predict: it may lead to frustration, but may also increase desirability and status. Some lessons can be learned from similar contention for ATMs and information kiosks, although other options apply to our scenarios. In any case, it is important to provide mechanisms to deal with such contention. One mechanism, which we have implemented, is the ability to "camp" on a shared FD, and receive notification on your MD when your turn has come. How this would work in practice, in a dynamic zone with people moving about, needs to be studied. Another option is to support sharing, where the large screen can be physically partitioned, or mashed-up information on the same zone map can be time-multiplexed. Another point to consider is that displaying mashup results on a shared medium can provide fertile ground for planned and ad-hoc social interaction.

We predict that as the perceived value provided by UDM increases, so will users' tolerance for contention. Anecdotal evidence supporting this prediction is the adoption of self-checkout registers in supermarkets and other stores in the US, where we have observed that people are now lining up for self-checkout registers as they are for registers with a human cashier.

5.4 Flexibility vs. precision

In UDM we want to empower users to easily specify what is mashed up and how it is displayed, while enabling providers of data services and zone maps to create a wealth of artifacts that interoperate seamlessly. There is a tension between these two goals, since diversity in the types of services and

maps would intuitively increase the burden on the user for specifying their intent. Our design advocates a middle-of-the-road approach, whereby data services can provide rich information, but need to adhere to a particular structure, and that structure is leveraged in populating the mash detailer. Our design encourages service developers to make their schemas as precise as possible (for example, using enumerations where possible rather than free-form strings), since those schemas are exposed to users in the mash detailer interface. As part of our related Celadon work, we have built tools for helping developers create such structured data services.

The other place where we need to reconcile flexibility and precision is for the mash maker, in both the joiner and transformer. As indicated in Section 2.6, XPath expressions can be used for precise specification of matching criteria in the joiner. However, such expressions would need to be specific to the schemas of the two input services matched. This creates a dependency between services and impedes their independent construction. Therefore we need to support generic rules for matching, which may be less precise, but are advantageous from a systems management perspective. Our current implementation does a fuzzy comparison of field names and values (e.g., ignoring capitalization and white space), and associates matching conditions with the zone map's directories.

An avenue we are exploring is use of knowledge-based technologies for smarter matching of field names and values (e.g., matching a "phone" field with a "telephone number" field). This is related to the thread of Celadon research on the use of ontologies for semantic representation of information about zones (see [27] for related work in this area).

The same issue, of dependency between services, arises when using XSLT templates in the transformer, since the templates may rely on both input formats as well as the output format. Our implementation imposes a particular output format (for the working set), known to the mash renderer, and uses service-independent rules for assembling an output record from the result of the joiner.

5.5 Leveraging user preferences

An alternative approach to specifying mashup details via a GUI is to infer such details from the user's preferences, which may be stored on the MD or available through a data service. According to this approach, the mash detailer component would contain an inference agent that relates user preferences to the selected data service. This, again, is a matching problem, similar to that faced by the joiner component of the mash maker: preferences are stored as property-value pairs, and the inference agent would need to locate preference properties that match fields of the data service record. For instance, in our restaurant locating example, the inference agent may find that the user has a "cuisine" preference, and would apply its value to the query. Thus, for example, a vegetarian user may, by default, get to see by default those restaurants that serve vegetarian food, without having to repeatedly state that explicitly in the mash detailer GUI.

A hybrid approach may be most advantageous, where the inference agent extracts preference values to pre-populate fields in the mash detailer GUI. The user may then simply hit "submit" to use his personalized default search criteria, or may chose to interact with the GUI to further refine or modify the query parameters. The mash detailer GUI can be further enhanced to save the entered field values as user preferences for future reuse.

Finally on this topic, as became apparent in our COEX mashup experience, treatment of language preferences is very important in this application area.

5.6 Scalability

Assessing the scalability of our architecture is an important part of our future research. Scalability has multiple facets in our system, some of which have already been mentioned. In *interaction scalability* we need to model and study how competition for shared resources (facility displays) affects usage and adoption. In *performance scalability* we need to measure system responsiveness as numbers of users, devices, and services increase. In *diversity scalability* we need to study how well our UDM design and current implementation can accommodate a large variety of users, data services, zone maps and usage scenarios.

5.7 Mobile mashups

An alternative approach for UDM is to present the mashup result on the mobile device itself, thereby keeping the interaction contained to the user's personal space. Compared to our method, this general approach alleviates competition for shared resources and simplifies privacy issues, but is disadvantageous with respect to display quality and social interaction. From a technical viewpoint the main challenge is to provide a mobile platform for visualizing zone maps with superimposed data. Some existing systems that do so for geographical maps include Google Mobile Maps and a variety of specialized GPS devices.

6. Related work

An broader overview of Celadon is given in [23] and [13], and an in-depth exposition of Celadon's RESTful data services design and usage is in [14].

In the introduction we explain how our work relates to situational applications [25], [9], [21], and to Web sites that let the user specify the features of the data to be mashed up (such as [26]). Another kind of system that provides user interface features for selecting the mashup data is Google Earth [5], which is primarily a viewer for layered XML information (in a format called KML). KML documents contain descriptions of both the data (map features) to display and the user interface elements for letting the user control what is displayed. Although the artifacts are declarative (all XML), here too the data to display and the interface controlling it are explicitly authored, as compared with UDM where they are generated at runtime.

The main competing approach to UDM is document-based search, where a search engine searches over an indexed corpus of many documents, and presents a ranked list of the best ones in response to user-provided search criteria. There is a growing body of work on mobile Web search (such as [10], [24], [22], [30] to name a few). An unstructured variant of document-based search is text-based search, in which documents are retrieved based on a textual search term. A structured variant employs classification for the corpus, where documents are retrieved based on nested classifiers specified incrementally by the user. This variant can narrow the search more precisely than free-form text search, but requires multiple round trips with the server. An example of an intermediary approach is Yahoo's Search Assist [29], which dynamically computes and displays suggested terms for refining the current search term (that can be viewed as classifiers). Documents can also be retrieved with a context, such as location. For example, Google Local Search uses the location information from a map shown in the browser to find nearby businesses for whom a document was created with structured location information [6].

In comparison, in UDM the "documents" are data-service records. A top-level classification of documents is done through their separation into different data services, and the user interface for that level of selection is the service explorer. However, further classification is done in a single round trip via the mash detailer (as opposed to iterative refinement) by inspecting the data schema. Mashing up the retrieved documents to produce a working set is required for bridging low-level location information with high-level data, as explained in Section 5.1.

An alternative that is interesting to contrast with UDM is the use of SMS for search [24], where the goal is to leverage a very simple mechanism that is commonly available to provide useful functionality, as opposed to rich visual information.

Newman et. al. [18] also address the issue of having mobile device users interact with other devices and services in their environment. Their focus is on enabling users to create, through their mobile devices, flexible associations with devices and services so as to achieve tasks that were not preprogrammed. According to their user studies, presenting spurious components caused confusion, whereas stricter guidance helped users achieve their tasks more efficiently. This finding is in line with our UDM design philosophy, in which the basic process is quite restricted (locate map, service, display, and submit data query), and the flexibility results from having multiple rich data sources.

Interesting work has been done in using constrained devices for displaying rich mapping information. Baudisch and Rosenholtz address the necessity of dealing with off-screen locations on a map [2]. Kray et. al. evaluate different means of presenting route information on mobile devices, from spoken instructions to rich visualizations . Cheverst et. al. [4] provide contextual information (a tourist guide) through mobile interaction. Their approach utilizes a nonstandard device that is larger than typical phones and PDAs, and can thus offer a richer experience. Cyberguide is another example in this space [1]. Our approach, in contrast, builds upon standard devices and platforms, leveraging facility displays for a rich experience. The use of multiple devices for display and control is advocated in [16]. Here we extend these ideas into public spaces.

Zone maps for interior spaces and data mashups on them are still at an early stage, but gaining popularity on the Web. Nearby Now [17], for example, provides text-based search for products in various shopping malls throughout the US. The response to a search query is a list of matching documents as well as a mall map with markers for each of the matching stores.

7. Conclusion

User-driven mashups are a promising new approach that leverages device symbiosis between mobile devices and facility displays to support effective gathering of information in public spaces. We have presented details of this technique, and

explained how it compares favorably with alternatives in this space. Experience with our initial prototype for a mall scenario has led to an engagement in the Ubiquitous City in Korea. We plan to use this opportunity to further refine this work.

8. Acknowledgements

We thank Sean Lee and François Huaulmé for their early contributions to the design and implementation of UDM. This work is partially supported by the Institute of Information Technology Assessment (IITA) and the Ministry of Information and Communications (MIC) of Republic of Korea.

9. References

[1] G. Abowd et. al., "Cyberguide: A mobile context - aware tour guide", *Wireless Networks* 3(5), Oct. 1997, pp. 421-433.

[2] P. Baudisch and R. Rosenholtz, "Halo: a technique for visualizing off-screen objects", *proc. ACM CHI 2003,* pp. 481-488.

[3] S. Berger et. al., "Using Symbiotic Displays to View Sensitive Information in Public", *proc. IEEE PerCom* 2005, pp. 139-148

[4] K. Cheverst et. al., "Developing a Context-aware Electronic Tourist Guide: Some issues and Experiences", *proc. ACM CHI* 2000, pp. 17-24.

[5] Google Earth: http://earth.google.com/

[6] Google Local Business Center: https://www.google.com/local/add/login

[7] Google Maps API: http://www.google.com/apis/maps/

[8] Housing Maps: http://www.housingmaps.com/

[9] Intel MashMaker: http://blog.programmableweb.com/2007/07/06/intels-mashmaker-coming-soon/

[10] M. Kamvar and S. Baluja, "A large scale study of wireless search behavior: Google mobile search", *proc ACM CHI 2006*, pp. 701-709.

[11] C. Kray et. al., "Presenting route instructions on mobile devices", *proc. 8th international conference on Intelligent user interfaces, 2003,* pp. 117-124.

[12] Lotus Expeditor: http://www-306.ibm.com/software/lotus/products/expeditor/

[13] S. McFaddin et. al., "Celadon: Delivering Business Services to Mobile Users in Public Spaces", IBM Research Report RC24381, Oct 2007 (submitted for publication).

[14] S. McFaddin et. al., "Modeling and Managing Mobile Commerce Spaces using RESTful Data Services", IBM Research Report RC24344, Nov. 2007 (submitted for publication)

[15] MQTT: http://www.mqtt.org/

[16] B. Myers, "Using Multiple Devices Simultaneously for Display and Control", *IEEE Personal Communications,* Oct. 2000.

[17] Nearby Now: http://nearbynow.com/

[18] M. Newman et. al., "Designing for Serendipity: Supporting End-User Configurations of Ubiquitous Computing Environments", *proc ACM DIS* 2002, pp. 147-156.

[19] OSGi: http://www.osgi.org/

[20] Planograms: http://en.wikipedia.org/wiki/Planogram

[21] QEDWiki: http://services.alphaworks.ibm.com/qedwiki/

[22] K. Rodden et. al., "Effective Web Searching on Mobile Devices", *proc. 17th Annual Conf. on Human-Computer Interaction, 2003.*

[23] M. Rosu et. al., "Celadon: A Novel Architecture for Symbiotic Computing", *proc. Intl Symp on Ubiquitous Computing Systems (UCS) 2006.*

[24] R. Schusteritsch, S. Rao and K. Rodden, "Mobile search with text messages: designing the user experience for Google SMS", *proc. ACM CHI 2005,* pp. 1777-1780.

[25] Situational Applications: http://en.wikipedia.org/wiki/Situational_application

[26] Trulia: e.g., http://www.trulia.com/FL/Miami/

[27] X. Wang et. al., "Semantic Space: An Infrastructure for Smart Spaces", *IEEE Pervasive Computing,* 3(3), Jul-Sept 2004.

[28] Yahoo Pipes: http://pipes.yahoo.com/

[29] Yahoo Search Assist: http://tools.search.yahoo.com/newsearch/searchassist

[30] X. Xie et. al., "Efficient Browsing of Web Search Results on Mobile Devices Based on Block Importance Model", *proc. IEEE PerCom 2005*, pp. 17-26.

[31] XPath: http://www.w3.org/TR/xpath20/

[32] XSLT: http://www.w3.org/TR/xslt20/