User-Defined Visual 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. The Celadon framework, previously developed by our team, 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-Defined 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.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures, patterns.*

General Terms
Design, Human Factors, Experimentation, Management.

Keywords
Mashups, Mobile Computing, Services.

1. INTRODUCTION
Consider the scenario where you are at a mall shopping for some clothes. Next to you, a video display is showing advertisements that don't concern you. You find a nearby directory map and look for stores of interest and also for restaurants (it is getting close to lunchtime…). For each establishment of interest you need to laboriously search for its location on the map.

Mobiquitous 2008, July 21–25, 2008, Dublin, Ireland.

Now consider the following twist. You use your personal mobile device to request information, whereupon an interactive mall map appears on the large display next to you. Through your mobile device you ask to locate clothing stores, then change your mind and choose restaurants. Your mobile presents an interface for specifying restaurant-relevant attributes, such as cuisine type and price-range. Once specified, the matching restaurants are highlighted on the map on the large display. Further interaction yields more details on selected restaurants, and also lets you book a reservation.

This paper describes a system for enabling the second type of scenario. Based on simple user interactions, the system locates data sources, queries them, and visually composes their outputs. Our system extends Celadon [19] – a framework we created for delivering zone-based services through personal mobile devices and public infrastructure devices. The work presented here supports ad-hoc, loosely structured activities, where 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 with current technologies is via text-based search such as used by the Google maps service for local business search [6]. 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 approach where the system helps guide the user in what to specify.

Addressing locations in indoor spaces raises interesting challenges. Geographic mapping software relies on the widespread existence of geocoding and reverse geocoding systems to convert between street addresses that humans understand to a precise mathematical representation of a location and vice-versa. No analogous system has been developed for indoor spaces yet. Also, indoor spaces may have several floors, necessitating 3D coordinates, and typically cannot leverage GPS technology. Thus we had to devise an approach that works for indoor maps.

Mashups are a prevalent paradigm for information composition on the Web. A mashup combines data from more than one source into a single integrated application that is not provided by any individual source. Visual mashups show data within a meaningful visual context (such as on 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] or markup [11]. Tools such as Yahoo Pipes [32] and QEDWiki [25] enable easy creation of mashups, also by non-programmers. Such tools create situational applications [28], which are easily constructed, lightweight, and serve specific purposes. The purpose of such tools is to create applications that are subsequently run.

In contrast, in the fast-paced scenario described earlier, 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 automatically, based on minimal input from the user. We call them *user-defined mashups*. Whereas the data sources are pre-specified in conventional mashups, user-defined mashups allow data sources to be specified and composed at runtime, based on user input. Such a partitioning allows composition of data sources published by different entities, e.g., cuisine information can come from a restaurant service and the restaurant location from a directory service. In such a world, GUIs are built from schemas that describe data streams and allow the specification and creation of composite data streams that can then be rendered on a map. Another central feature is that this new breed of mashups does not require programming by a human..

The key contribution of this paper is a new methodology for automated data composition. User-defined mashups enable delivery of mapped zone information to mobile users in interactive public spaces, addressing the following challenges:

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

Enhancements to our current implementation will 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.

Our work is distinguished from existing Web sites, such as Trulia [30], which let the user easily specify the desired features of the information to be displayed (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.). 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 logic for composing data from multiple services 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, and outline future directions. We close the paper with a discussion of related work and conclude.

2. APPROACH AND ARCHITECTURE

This section describes our approach for User-Defined Mashups

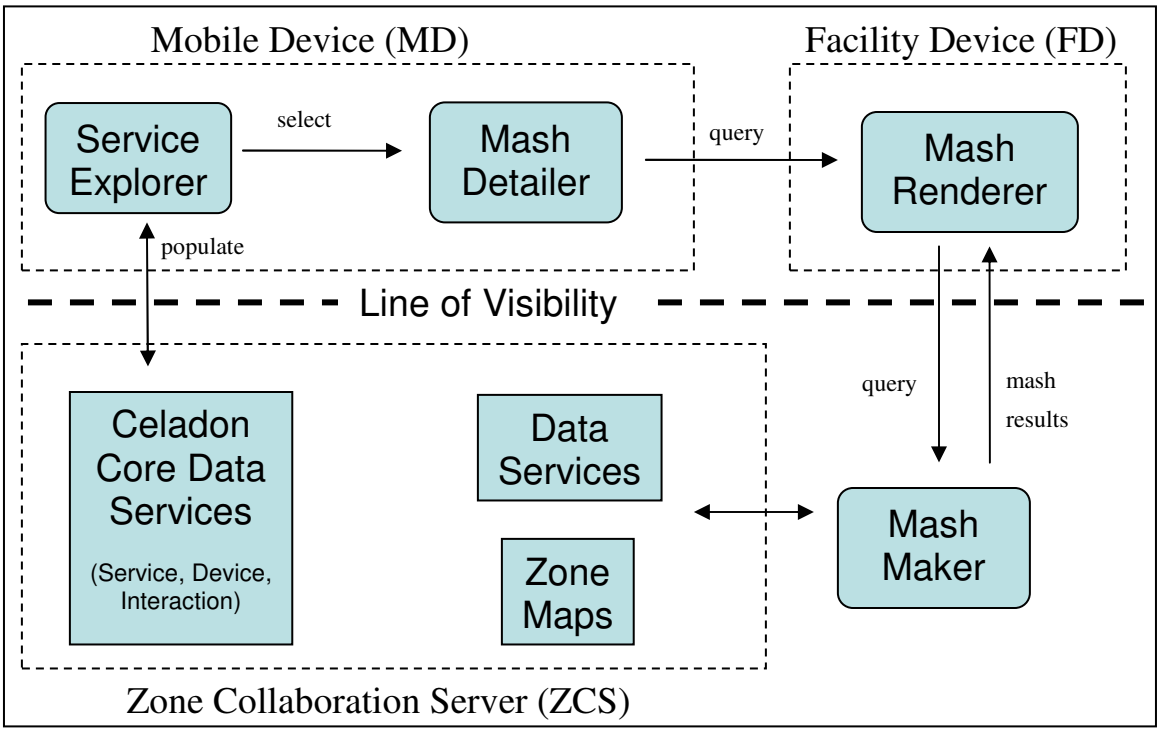

Figure 1: User-Defined Mashup Architecture. Components above the line of visibility are seen by the user.

(UDM) and the system architecture. We briefly describe the pertinent parts of the existing Celadon architecture [19], [16], followed by a detailed description of the new elements supporting UDM.

2.1 Celadon Architecture

2.1.1 Distributed Architecture

Celadon's 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, services, and its associated members (MDs and FDs).

2.1.2 Data Services and Zone Maps

RESTful data services are a central building block in Celadon's software architecture [16]. Data is retrieved from such a service by passing it an appropriate query via an API call or an HTTP request. The ZCS keeps track of members, services, locations and interactions in the zone 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.

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: they provide the schema for their records as well as for queries against those records. A given data service may support multiple record types. As a convention in our current implementation, a record contains zero or more *fixed fields* and zero or more *attribute assertions*. Fixed fields are typically used for a singleton value, such as name. Attribute assertions are more free-form, and can be used when a record contains multiple values, such as phone number (when an establishment has several phone numbers). Each attribute assertion is an SVO triple – <subject,verb,object> – where the subject is an attribute name, the object is a value, and the verb is a comparator. For example, <cuisine."=","Korean"> denotes a Korean cuisine, and (floor,">",5) refers to 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.

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 being mashed up.

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 metadata with the physical locations. Such metadata may be logical location information or other information associated with that physical location such as a store's name or phone numbers. 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.

2.2 UDM Process and Architecture

The central operation in UDM is to locate data sources of interest, from them obtain relevant zone data and show the results 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 available FDs. 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 the new 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* computes the mash results for presentation on the renderer (it may run anywhere and therefore is drawn outside any dotted rectangle).

2.3 Service Explorer

The starting point for the UDM process is to present available services and resources on the user's MD. This is done through the service explorer, which queries core data services on the ZCS to retrieve listings of devices, zone maps, and data services. A listing of large facility displays 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. These queries may be tailored to the user's preferences or context (such as location), for example retrieving only displays in the user's 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 and zone map, and then pick a data service for mashup. Once a zone map is selected, the system limits the choice of data services to ones compatible with the selected map. For example, informational services for the entire shopping mall may be separate from ones for a department store within the mall.

2.4 Mash Detailer

Specifying parameters for the chosen data service is done on the mobile device via the mash detailer. Its design relies on the regular structure of Celadon data services, as it presents an interface for creating data service queries. 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.

A possible enhancement to having the user enter the mashup details is to infer them from the user's preferences. This approach, as yet unimplemented, is discussed in Section 5.5.

2.5 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 sends the query created by the mash detailer to the selected data service; it then takes the set of records returned by the data service and composes them with records from the zone-map directory, 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.

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.

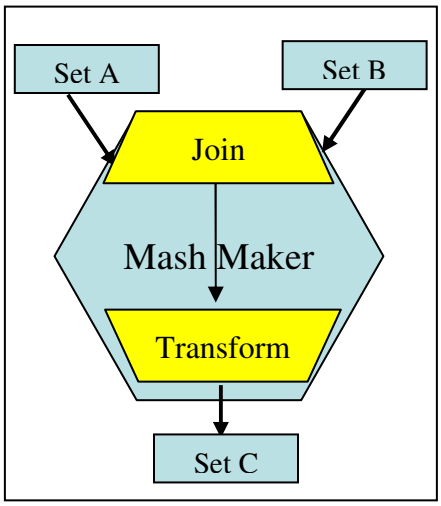

Figure 2: Mash maker design

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. Matching rules can be modeled with XPath expressions [35], which 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 records are transformed to a given target format. Transforms can be modeled with XSLT templates [36].

Architecturally, the mash maker 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. Our current implementation uses more limited modules, for reasons discussed in Section 5.4.

2.6 Mash Renderer

The mash renderer displays the working set superimposed on the zone map on the FD. 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 and relays the data query to the mash maker (architecturally it is simpler to have the mash maker deal directly with the mash renderer). The mash renderer takes the working set returned from the mash maker and visually superimposes it on the zone map. 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 to be notified of any underlying data changes. If a change occurs, the mash renderer re-invokes the mash maker to obtain an updated working set, and refreshes the display.

2.7 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. We 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 (e.g., maps of entire floors of the mall), including 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 needs to update the relevant zone maps accordingly. Zone administrators also host the core data services of the zone, including one listing available services: all data services and zone maps 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 their data services with the zone.

Providers of physical services (e.g., 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 benefits, 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[15], which extends the Eclipse Rich Client Platform (RCP). The Expeditor runtime uses OSGi [23], which is a

container for Java components called bundles. On the FD we leverage the embedded browser feature of RCP.

Service Explorer: Figure 3 shows a service explorer screen shot, listing a facility display, a service, and a user's mobile device. It provides a view of available resources in the zone, leveraging Celadon's dynamic service discovery feature. Here the mashup viewer 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 a sample restaurant info service.

Figure 4: Mash detailer screen shot

The fixed fields are: Location ID, Operating Hours, and Name. 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., "Cost" in this example), the verb and object fields are populated. Clicking the small arrow button on the right adds an assertion. The attribute assertions are collected and displayed in a text area, where they can be edited manually by the user if desired. 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. The example query shown in Figure 4, selects 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 their values agree for all the named fields. 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 than 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: The 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 mobile device and zone collaboration server. Its code structure is shown in Figure 5.

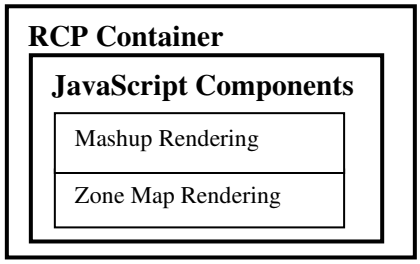

Figure 5: Mash renderer code structure

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 for zone administration, where zone managers can monitor zone status and activity. We also built 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. In these displays the mashed information is shown as informational markers on the zone map.

Performance considerations: to help achieve good performance, especially on the MD, we have used lightweight components. For efficient messaging we use MQTT [17], which is optimized for resource-constrained 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. 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 devices available in the market,

data services and zone maps. This section describes two evaluations of UDM: their setup, results and lessons learned. The first evaluation was in a lab demonstration setting, and the second was in a public demonstration setting; broader evaluation of 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 shopping mall attached to the COEX (Convention and Exhibition Center) in Seoul, South Korea. 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 due to language issues. An informal survey of local visitors revealed that they thought they could benefit from user-defined 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 books, sports goods, etc), and some additional details about the type of goods they are interested in.

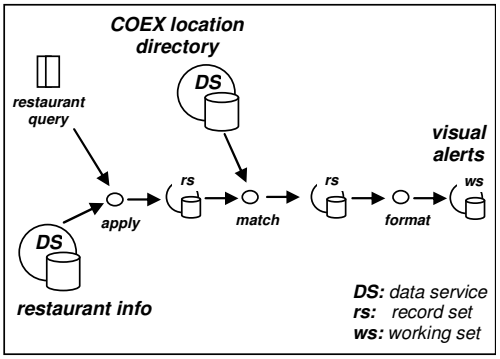

Figure 6: Data flow for mall mashup

Figure 6 explains the data flow involved in mashing up restaurant data on a 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 (transformed) 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, 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 deployed two different MD types in IFEZ: Windows Mobile PDAs (HP iPAQ) running Lous Expeditor, 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 Lotus Expeditor.

We implemented three IFEZ zone maps, for showing attractions, a building guide and organization chart. A search for attractions shows 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 zone maps 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 7. 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 connecting 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. Selection of a zone map displays it on the associated FD, and mashed up data is computed and displayed by following the UDM process described in Section 2.2. Further interaction with the displayed data can be done through the MD or directly on the FD.

4.2 Evaluation Results and 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 function through a rich albeit small display, 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 near-field interaction such as touching or swiping. Due to 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 visually stunning or 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. We are considering evolving our system to use Adobe Flex 3 Builder™ or Microsoft Silverlight™ on the FD for a richer multimedia experience.

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 sub-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 [24], 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.

Producing this style of mashup involves (1) importing zone map data from preexisting sources, (2) identifying key zone map elements as output targets for mashups, and (3) constructing 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 (which we have under development) that allow the designer to browse the imported diagram using the mouse, selecting only key elements to be used as mashup targets. Alternatively, an image may serve as the underlying representation, with invisible bounding polygons as the mashup targets. Step (3) involves construction of data elements that 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 in the mashup process.

The three informational data services in the COEX demo were developed in part using the Celadon data tools, which take as input a service record schema and generate various code artifacts, including a JDBC-based service implementation and specialized JavaScript for the mash renderer.

Since there is an overhead for each data service implemented (e.g., using the data tools, maintaining RDB tables), we sought 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.

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 is 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 technique 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 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 JavaScript client that, similarly to the mash detailer, inspects the service schema at runtime to produce an interface that is tailored to that data service.

Figure 7: Computed mashups –restaurant info in COEX mall (left), floor-plan info in IFEZ public demonstration (right)

5. DISCUSSION AND FUTURE 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 similar schemes for arbitrary zones. We solved 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 broad benefit 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, 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 involves both the user's personal mobile device and large shared displays. Thus the process is part private and part public, and it is important to keep sensitive information private. 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 is a factor for adoption, and should be studied in actual deployments. In more elaborate scenarios, where the user query is constructed based on user preferences or other personal information (e.g. a shopping list), more care needs to be taken to keep 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 the IFEZ deployment we 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, and 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 serendipitous social interactions.

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 to mash up and how to display it, 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 that is leveraged by the UDM components. 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.

We also need to reconcile flexibility and precision is in the mash maker, in both the joiner and transformer. As indicated in Section 2.5, XPath expressions can be used for precise specification of matching criteria in the joiner. However, such expressions may need to be service-specific, and may create dependencies between services. 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, and associates matching conditions with the zone map's directories. We are exploring the 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 part of an investigation on the use of ontologies for semantic representation of zone information (see [31] for related work in this area).

Similarly, 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 would contain an inference agent that relates user preferences to the selected data service. This 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 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 only restaurants that serve vegetarian food, without having to 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: 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 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 variant of UDM is to present the mashup results on the mobile device itself, thereby keeping the interaction contained to the user's personal space. Compared to our method, this approach alleviates competition for shared resources, improves mobility and simplifies privacy issues, but is disadvantageous with respect to display affordances 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 and of its RESTful service design and usage are given in [19] and [16] respectively.

In the introduction we explain how our work relates to situational applications [28], [9], [25], and to Web sites that let the user specify the features of the data to be mashed up (such as [29]). 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 [11]). 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 an 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], [27], [26], [34] 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, but requires multiple round trips with the server. An example of an intermediary approach is Yahoo's Search Assist [33], 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 which 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 via 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 [27], 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. [22] 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, where the process is streamlined (locate map, service, display, and submit data query), and the flexibility results from having multiple rich data sources. Kruppa and Krüger [14] explore various forms of combined simultaneous interaction with PDAs and large displays. They classify systems according to the display arrangement – separated (PDA and large display serve different tasks), integrated (PDA shows part or all of large display), or extended (the two devices show different parts or aspects of the same picture) – and focus of interaction (PDA, large display, or both).

Significant 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. [12] evaluate different means of presenting route information on mobile devices, from spoken instructions to rich visualizations. Interestingly, follow-on work [13] investigates the use of large adaptive displays embedded in the space for the same task (navigating the space). 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 [1] is another example in this space. Our approach, in contrast, builds upon standard devices and platforms, leveraging facility displays for a rich experience. Symbiotic advertising on large displays by utilizing user preferences from users' mobile devices is explored in [20]. The use of multiple devices for display and control is advocated in [18]. 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 [21], 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. A class of mashups for use in the enterprise, such as OrgMaps [29], for insights into organizational structure, is beginning to appear on the horizon.

7. CONCLUSION

User-defined mashups leverage device symbiosis between mobile devices and facility displays to support effective gathering of information in public spaces. We have presented details of this novel technique, and explained how it compares favorably with alternatives in this space and advances the state of the art in automated mashups. 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. ACKNOWLEDGMENTS

We thank Sean Lee and François Huaulmé for their early contributions to the design and implementation of UDM, and appreciate Jonathan Munson's feedback on the manuscript. This work was 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://mashmaker.intel.com/

[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] KML: http://code.google.com/apis/kml/documentation/

[12] C. Kray et. al., "Presenting route instructions on mobile devices", *proc. ACM IUI, 2003,* pp. 117-124.

[13] C. Kray et. al., "Adaptive Navigation Support with Public Displays". *proc. ACM IUI 2005*, pp. 326-328

[14] M. Kruppa and A. Krüger, "Concepts for a combined use of Personal Digital Assistants and large remote displays", *proc. SimVis 2003,* pp. 349-361.

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

[16] S. McFaddin et. al., "Modeling and Managing Mobile Commerce Spaces using RESTful Data Services", *proc 9th Intl. Conf. Mobile Data Management*, May 2008, pp. 81-89.

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

[18] B. Myers, "Using Multiple Devices Simultaneously for Display and Control", *IEEE Personal Comm.,* Oct. 2000, pp. 62-65.

[19] C. Narayanaswami et. al., "Device Collaboration for Ubiquitous Computing", *IPSG SIG Technical Reports*, ISSN 0919-6072, Japan, Vol. 2005, No. 60, pp 7-12.

[20] C. Narayanaswami et. al., "Pervasive Symbiotic Advertising", *proc HotMobile, 2008.*

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

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

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

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

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

[26] K. Rodden et. al., "Effective Web Searching on Mobile Devices", *proc. 17th Annual Conf. on Human-Computer Interaction, 2003*, pp. 281-296.

[27] 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.

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

[29] D. Soroker et. al., "Organizational maps and mashups", IBM Research Report RC 24551 (May, 2008). Submitted for publication.

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

[31] X. Wang et. al., "Semantic Space: An Infrastructure for Smart Spaces", *IEEE Pervasive Comp.,*Jul-Sept 2004, pp. 32-39.

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

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

[34] 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.

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

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