

Personalized Location-Based Services

Yiming Liu
School of Information
UC Berkeley
yliu@ischool.berkeley.edu

Erik Wilde
School of Information
UC Berkeley
dret@berkeley.edu

ABSTRACT

Location-Based Services (LBS) are based on a combination of the inherent location information about specific data, and/or the location information supplied by LBS clients, requesting location-specific and otherwise customized services. The integration of location-annotated data with existing personal and public information and services creates opportunities for insightful new views on the world, and allows rich, personalized, and contextualized user experiences. One of the biggest constraints of current LBS is that most of them are essentially *vertical services*. These current designs makes it hard for users to integrate LBS from a variety of service providers, either to create intermediate value-added services such as social information sharing facilities, or to facilitate client-side aggregations and mashups across specific LBS providers. Our approach, the *Tiled Feeds* architecture, applies the well-established, standard Web service pattern of feeds, and extends it with query and location-based features. Using this approach, LBS on the Web can be exposed in a generalized and aggregation-friendly way. We believe this approach can be used to facilitate the creation of standardized, Web-friendly, horizontally integrated location-based services.

1. INTRODUCTION

With the advent of smartphones and other sophisticated technologies for users to interact with Web-based services, *Location-Based Services (LBS)* [7] have seen a surge in popularity. While some utilization of spatial information has been in use on the Web for a long time (popular examples are mapping services such as *Google Maps*, or localized services in search and directory services), the increasing popularity of mobile devices and localized services have spawned specific services around location. Some of these services are Web-based (popular examples include the recent startups around personalized location tracking such as *Foursquare* and *Gowalla*), and others are local (popular examples are the browser-based *W3C Geolocation API* [19], and location

services provided by non-Web platforms such as *iOS*, *Android*, *Symbian*, or *Maemo/MeeGo*). Facebook's recently launched *Places* functionality introduces location concepts into the Facebook platform, but follows the platform's general strategy to try to establish it as the central platform for all users and information exchanges.

In all, consumer LBS promise personalization of services and contextualization of information [21]. As a classical example, a request for local weather conditions might be automatically personalized for where the client is currently located. In a more modern "Web 2.0" instantiation, this might consist of notifications of when one's friends are within a certain vicinity, or location-specific notes for friends to discover (e.g. "the café here makes great lattes but terrible sandwiches"). They may even integrate multiple types of information; Foursquare, for example, not only provides information about the movements of one's social circle, but also tips from the general public on businesses nearby.

While LBS are the foundation of many popular applications (both Web-based and otherwise), there is a surprising dearth of LBS that exist in a truly open and Web-friendly way. Consider the simple but illustrative example of Web-linking to a physical location, so that visitors can locate it and obtain additional services (such as directions or tourism information). In many cases, Web sites simply embed maps from mapping services such as Google Maps. This tightly couples the Web site to that particular service provider. Switching map providers (say, to Yahoo, Bing, or OpenStreetMaps), or to load additional data (say, navigation) from some other source, is non-trivial. A more robust way, using standard Web technologies and architectures, would provide a declarative notion of location information and allow clients to integrate services as needed [8]. The recently standardized *geo* URI scheme [10] is a first step in that direction. This URI scheme supports URIs using lat/long coordinates, so that a supporting client could understand that *geo:37.871384,-122.258285* points to a physical location. The client can then invoke its preferred set of services with that location, such as using a mapping service for visualization, a navigation service for routing, Yelp business listings for nearby tourism information, Facebook Places for friends in that area, and so on.

In a similar vein, many location-based applications are designed and implemented in a vertical style, integrating a specific use case. The typical LBS design scenario is that developers pick a set of back-end data sources (say, a database, or an API such as Flickr or Twitter), a mapping service (Google Maps, mostly), and then write specific application

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

iConference 2011, February 8-11, 2011, Seattle, Washington, USA.
Copyright 2011 ACM 978-1-4503-0121-3/11/02 ...\$10.00.

code that integrates with each of their data sources. This, in turn, tightly couples the developer’s new LBS application to the chosen platforms and data source APIs. In simple cases, well-established formats such as KML [15] can be directly retrieved from a data source or supplied to a mapping service. In more likely scenarios, developer code must shuttle data from each back-end, convert to some intermediate data structure or form, detect map views and viewports on the front-end, and dynamically produce visualizations or mashups. In all cases, this creates tight couplings, but especially between the application and its datastores. For N services used by the application, N custom integrations must be made. This makes creating applications from multiple LBS difficult at best, and discourages the creation of services that make integrated use of available LBS on the Web. Instead of drawing or linking to data from multiple service providers, as per the spirit of the Web in general, individual services grow vertically to encompass more use cases, or applications take on additional complexity to interact with all of these services.

In this paper, we discuss our approach for an LBS architecture from the perspective of service design and horizontal service integration. The main goals of our approach are to produce a LBS design framework that is usable across a wide variety of LBS scenarios, is based on well-established and widely used Web technologies, and provides open and standardized access to location services. Our approach, called *Tiled Feeds*, can be used to create LBS that are open and accessible to basic Web clients, and can be easily integrated and remixed with other LBS to provide rich, personalized views and services. This approach is also an ideal starting point for building LBS that can be used across a variety of mobile delivery platforms, either as Web-based applications designed for mobile Web browsers, or as native applications being backed by Web-based services.

2. RELATED WORK

The *Open Geospatial Consortium (OGC)* has defined *Web Map Service (WMS)* [13] and *Web Feature Service (WFS)* [14] as two standards for accessing map imagery and map features using Web services. Both of these standards are based on SOAP, which means that instead of exposing imagery and geospatial features as Web resources, they expose functions that can be remotely called. While SOAP is a well-established standard, it is a service protocol that uses the Web and HTTP as a transport layer, and is not truly part of the Web in particular. A more Web-friendly, RESTful approach, on the other hand, exposes information in the interlinked fashion of Web resources [5]. Clients thus do not need to support SOAP or the specific set of functions defined by WMS/WFS. We claim that the decision between RPC-style Web services and RESTful Web services [18] should be based on the expected use cases, and that map imagery and even more so map features should be provided in a way which allows “serendipitous reuse” [22]. At a pragmatic level, SOAP is not supported natively by many development platforms, while basic HTTP clients and feed readers are often part of platforms’ core functionality.

The idea of tiled access to map information in general (which underlies our feed-based model for LBS as discussed in Section 3) has been proposed very early and goes back to the approach of using *Quadtree* [20] structures for organizing spatial data. Popular Web mapping services such as

Google Maps are using tiled access to map imagery for scalability reasons, so that the vast amount of map imagery can be efficiently stored, served, and cached. Beyond these specific implementations, the *Tile Map Service* [16] proposes a general scheme for how services can expose map imagery through a tiling scheme. However, APIs for location-based services such as Google Maps or Flickr do not use the tiling concept to organize and deliver spatial information. Proprietary APIs for location-base services are typically designed in an ad-hoc fashion, with different types of access capabilities, and different levels of functional sophistication. The simplest instantiations, such as Flickr, allow queries by point and radius, while other services allow user-specified bounding boxes and maybe even operations on these shapes.

To the best of our knowledge, no scheme so far has been proposed that combines the tiled model of spatial information representation with a RESTful architecture for accessing the resources organized in this fashion.

3. RESTFUL LBS WITH TILES

The main starting point for our LBS architecture is *Web Architecture* [6] and the underlying architectural style of *Representational State Transfer (REST)* [3]. We start from the assumption that spatial information can be represented as features with a spatial association, and that these features can be exposed through the most popular pattern of RESTful Web services today — feeds. By using *Atom* [12] as the established standard for exposing spatial information, we build on top of a well-defined pattern of how collections of information items can be delivered as Web-based data services.

3.1 Tiling

In the Tiled Feeds architecture, the world is recursively partitioned into smaller tiles using a quadtree-based [20] tiling algorithm. Each tile provides an Atom feed of location features. Each feature is represented as an Atom entry containing GeoRSS or KML geometries and metadata. The service is completed by using link relations [11] between tiles, which are used to navigate the various spatial dimensions. Clients moving left, right, up, or down, as well as zooming in or zooming out, simply request the service to expose information at a different tile or level of granularity. To initialize a map view which may contain several tiles, the Tiled Feeds server also provides convenience services (via a discoverable, self-describing protocol document) for finding the tiles necessary to populate a desired viewport or bounding box, at a given level of granularity. Since this model of interlinking feeds closely replicates the pattern of tiled map services (merely expanding the model from mere map imagery to arbitrary data contained within Atom feeds), we refer to the system as *Tiled Feeds* [9].

The main goal of this design of LBS is to implement a loosely coupled architecture [17], in which LBS providers can easily expose spatial data and associated services using Tiled Feeds, and clients have simple ways of using these services for aggregation, filtering, and republishing. Since this design exposes LBS as navigable hypermedia, it relies on the semantics of the fundamental data types involved (Atom for the feeds and KML for the content of the entries) and some additional resource relations (for navigating Tiled Feeds), but the design is easily and naturally open for extension, where LBS can add additional links to their feeds and/or

their data for exposing additional services, or they can provide alternative representations for entries — for example, exposing RDF so that LBS can be used in Semantic Web scenarios.

3.2 Personalization

Most scenarios will make it unlikely that all information available through a LBS provider should be retrieved, or could be reasonably managed by a client. A client rarely desires to see all features within a given area, and may in fact have precise requirements. For example, a hypothetical client application for an earthquake notification LBS service may only be interested in earthquakes of magnitude 4 or above, and only those which occurred recently.

We thus rely on feed querying [23, 24], which introduces an additional service model on top of the interlinked Atom feeds. Clients can discover a Tiled Feed’s query capabilities through a self-description mechanism, where each Tiled Feed exposes its query capabilities in a standardized and well-defined way.

Through the feed query capability, Tiled Feeds clients can easily create personalized views on location-based services and their outputs. The aforementioned earthquake client can simply read the self-description query schema for the earthquake LBS to discover the available parameters, and the user of the client can employ that knowledge to specify to the earthquake LBS to filter for earthquake magnitudes 4.0 and above. This query capability discovery process is standardized across all LBS based on Tiled Feeds, so no tight coupling is required.

In a service mashup scenario, query capability provides even more powerful personalization opportunities. A user interested in examining earthquakes in high-population density or high-risk locations, for example, can simply subscribe to separate LBS services — one for earthquakes, one for population density, and one for sensitive installations and facilities. She can specify query parameters offered by each LBS, creating a different mashup on the fly. The three-LBS combined view can express “earthquakes in densely populated counties and near hospitals,” or “earthquakes in rural regions with hydroelectric facilities.” This kind of “cross-service personalization” would of course require the client to combine (or *join*, in database parlance) the data from the individual LBS, but because of the unified interface for accessing each of those services this becomes a much more tractable task than in cases where each service provides its own specific API.

3.3 Information Services in Context

Many use cases involve specific interest in a location, rather than more exploratory notions of wandering around a map. For example, in consuming a LBS for news, or a social network LBS reporting friend locations, the user may only be interested in results in their immediate vicinity, or at particular locations of interest (e.g. “home,” “work,” “my son’s elementary school”). In these cases, the client would only be interested in a specific tile or set of tiles.

Once found, reading a particular tile is as trivial as reading any other blog or newspaper feed. Further, since the tiling scheme is canonical across all Tiled Feeds services, the client can read the same tile from multiple services to create a combined contextual view or information dashboard. Adding another source of data for that location is as easy

as giving a new feed URI to the client reader, and adding a new location of interest is as easy as finding a new tile (or set of tiles) to request.

4. IMPLEMENTATION

Our implementation of a Tiled Feeds server [9] is based on a few extensions of the Atom feed format (most importantly, adding link relations for linking Tiled Feeds, and adding query parameter descriptions to make the query capabilities of those feeds discoverable), and a convention on what to make available as content (KML containing the spatial representation of objects within the bounding box of the feed tile in question). Interaction with such a LBS essentially uses the same interaction pattern as using the map tiles from Google Maps when using the map viewer: moving the map means that neighboring feeds have to be accessed for the associated tiles, and zooming in or out means that the more or less detailed feed tiles have to be accessed.

Because of this design, individual Tiled Feeds can even be accessed with a generic feed reader, if the Tiled Feeds service decides to also include an HTML representation of the feed tile, and to make a feed available which does not require any query parameters and follows Atom’s default convention of using the `published/updated` time stamp as the primary sort key for providing access to entries. It is interesting to notice that while Atom has a strong bias towards timed information (entries must have a timestamp), it does not mandate that entries in a feed are sorted based on timestamp; this is just a convention that evolved because of the main use cases (news feeds) and the fact that without extensions, Atom has no standard mechanism for specifying a sort key.

The feed-based design also allows an easy transition into a write-enabled scenario by reusing the *Atom Publishing Protocol (AtomPub)* [4]. Following REST principles, AtomPub defines additional link relations (most importantly, the `edit` relation) and some other constructs to extend Atom’s read-only scenario to a service with the full spectrum of create/read/update/delete operations. For publishing new location-based data, standard AtomPub operations can be used to interact with all services based on Tiled Feeds to instantiate or modify data.

We have also implemented two prototype specialized clients that understand our Tiled Feeds extensions to Atom (Figures 1 and 2 show examples of how these clients work), as case studies of horizontal integration of Tiled Feeds services. In the first, we created an initial version of a Web-based Tiled Feeds reader, along the lines of Google Reader for ordinary feeds. In the second, we created an iOS version of the reader, but specialized for the mobile use case and running on a mobile platform instead of being browser-based.

4.1 Web Client

The Web reader is browser-based and designed for data exploration and personalized visualizations, and is capable of consuming a large number of location-based services using the Tiled Feeds architecture.

It operates similarly to ordinary feed readers, but is more powerful because of its support for spatial and query aspects supported by Tiled Feeds. Instead of being a passive consumer, the Tiled Feeds reader interacts with LBS services based on Tiled Feeds. As described in previous sections, the reader takes as input URI to Tiled Feeds LBS services, and

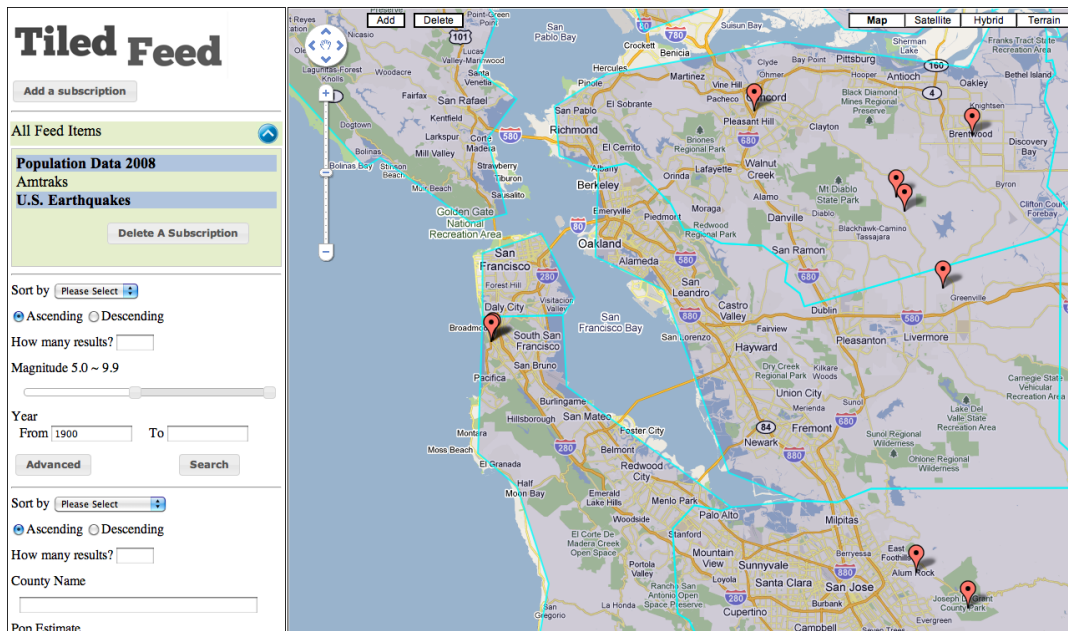


Figure 1: Prototype version of a Web-based Tiled Feeds reader, with three subscribed feeds and a mashup view of the SF Bay Area. Only two of the subscribed feeds are currently shown, with the “Population Data” feed showing polygons, and the “Earthquakes” feed showing placemarks.

uses the metadata documents located at the URI to find tiles and learn to about the feed’s query capabilities. As the user explores the map visualization, the reader retrieves the appropriate tiles and features at the desired zoom level to fill the map. It navigates between tiles using link relations, or make new requests for a new set of tiles as the user moves about the map. Possible query parameters for each service are provided in an automatically generated form on the left-side bar, and allows for fine-tuning and filtering of displayed data.

The reader is capable of AtomPub read/write operations over AtomPub-enabled Tiled Feeds services, allowing it to create or modify data contained within these services, or potentially perform administrative or editorial tasks. By clicking on a displayed feature, the reader is able to show the metadata associated with the feature, its geospatial geometry, and allow the user to edit these features at will.

It is important to note that all of this functionality is handled in a uniform, open way. No additional code or specialized adapters are required to interact with additional services. Users need only add a new URI for a compatible LBS, and can immediately start exploring the map. They can switch LBS data sources on and off from the interface, and create instant, personalized mashups of different data sources in any desired permutation.

4.2 Mobile Client

The mobile reader is designed for the typical mobile LBS use case: in-context data visualization and tracking. Unlike the more exploration oriented Web reader, the mobile reader creates views, based on data from single or multiple LBS, at specific locations. This allows for immediate application of LBS data based on the user’s current physical context, as well as allowing for continued monitoring of particular

locations of interest.

As with the Web reader, the mobile reader takes as input a set of LBS based on Tiled Feeds. As a second step, it asks the user to create a list of “Places” of interest. A Place can be a temporary construct (“Current Location”), or a more fixed one (`geo:37.8743,-122.2657`). For each Place, data from subscribed LBS will be shown for that location. The views are personalizable on a service-by-service, place-by-place basis, depending on the offered query capabilities. For example, a user might be interested in only earthquakes greater than magnitude 5.0 at one location, but earthquakes greater than magnitude 3.0 at another. She might be interested in a mashup of geotagged news articles and photos for one location, and Twitter postings and Foursquare check-ins at another. If Tiled Feeds compatible LBS are created for each of this purposes, each Place can be configured to show only the desired data when opened for viewing.

As with the Web client, the mobile client requires no additional code to handle new Tiled Feeds services. The user adds a new service and configures some query and display options. Once done, the new service is simply integrated into any current views.

5. LBS DESIGN

We discuss our primary considerations in establishing the Tiled Feeds architecture: the creation of a horizontally integrated, remixable architecture for LBS provision.

5.1 Horizontal Integration

We refer to *horizontal integration* as a design of an architecture which is useful and usable across a wide variety of scenarios from the same domain. Another way of referring to this goal often is called the *80/20* approach, or the *greatest common divisor*. There are various examples of

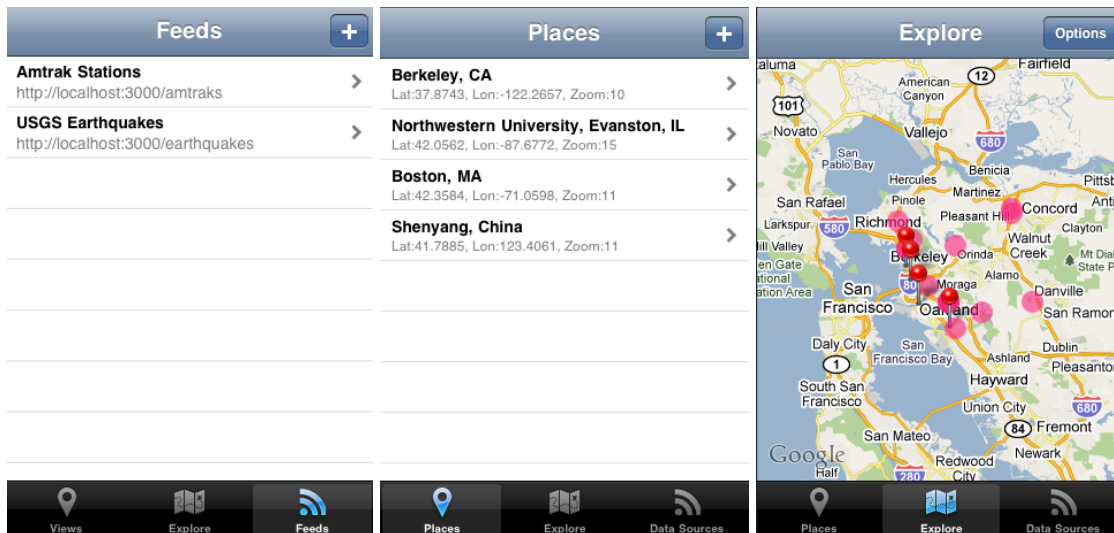


Figure 2: Initial version of a iOS-based Tiled Feeds reader, with two subscribed feeds, multiple subscribed places, and the resulting mashup view of the Berkeley, CA.

Web technologies where horizontal integration has created the opportunity to expose services to a much bigger selection of consumers than otherwise possible:

- *HTML* has implemented horizontal integration for human-readable documents. While its capabilities and sophistication as a document format are not especially powerful, the fact that it is interlinked and thus allows users to link to and explore related information has made it as uniquely successful as it is today.
- *Atom* has implemented horizontal integration for periodically updated information, and is produced and consumed across a wide variety of scenarios. Atom is very likely the most widely used “Web service” on a global scale, if the definition of a Web service requires some machine-readable information to be made available through such a service. Like HTML, Atom is not particularly sophisticated, but provides a very open and flexible platform for providing access to collections of information items.

A common pattern for successful horizontal integration platforms is that they are simple to understand and to use, that they are based on open standards and technologies, that they are extensible and thus can adapt to more specific user needs, and that they can evolve over time so that the integration platform slowly but naturally grows in sophistication over time.

Figure 3 shows how vertical and horizontal integration compare in the Google Maps scenario mentioned earlier (one provider for basic map imagery and a potentially large set of providers of LBS that are mashed up in the context of that map-based application). For vertically integrated services, special adapter code is required in the application, which tightly couples the services with that specific application. In the case of horizontally integrated services, no special adapter code is required, and LBS can be easily integrated because of a client-side framework supporting the generalized LBS model. Adding a new LBS in this scenario is as

easy as linking to it on the client side, so that the framework can start navigating the available resources.

Such a horizontally integrated LBS scenario should be easy to use and implement on any client platform supporting basic Web technologies. In the case of Tiled Feeds, the most important requirement is an HTTP client library, and an XML parser. Processing Atom is non-trivial and requires further abstraction atop a basic XML parser, but this effort already is mitigated by the availability of feed APIs in a variety of popular development environments. Specifically, the accessibility of services exposed this way is very good on any platform supporting Web technologies, not just for browser-based development of Web applications.

5.2 Mixing and Remixing LBS

The main objective for our design is to provide a low barrier to entry for exposing as well as for using LBS, and to provide a design that is in line with Web architecture and its loose coupling principles. While we use established Web technologies, we introduce some new link relations, and clients have to be aware of these relations to understand how resources (i.e., Tiled Feeds) are linked, and what those links mean. This pattern of linking Web-accessible resources with typed links is one of the core principles of REST, and we have developed the *Resource Linking Language (ReLL)* [2] for documenting and describing how resources are linked.

One of the main ideas behind ReLL is that in REST, service extension or composition is not an additional layer of complexity or description, it simply means adding links to resources, and maybe using new representations. In ReLL, service composition essentially means to take two services that are described in ReLL (i.e., for each of these services, it is described how the resources exposed by those services are interlinked), and then to create an additional set of resources that link those to sets of resources, and describe this new set of links connecting the two previously non-linked services with a third ReLL description.

The strength of this approach is that it is fairly easy to use LBS or composed services for building new services, simply

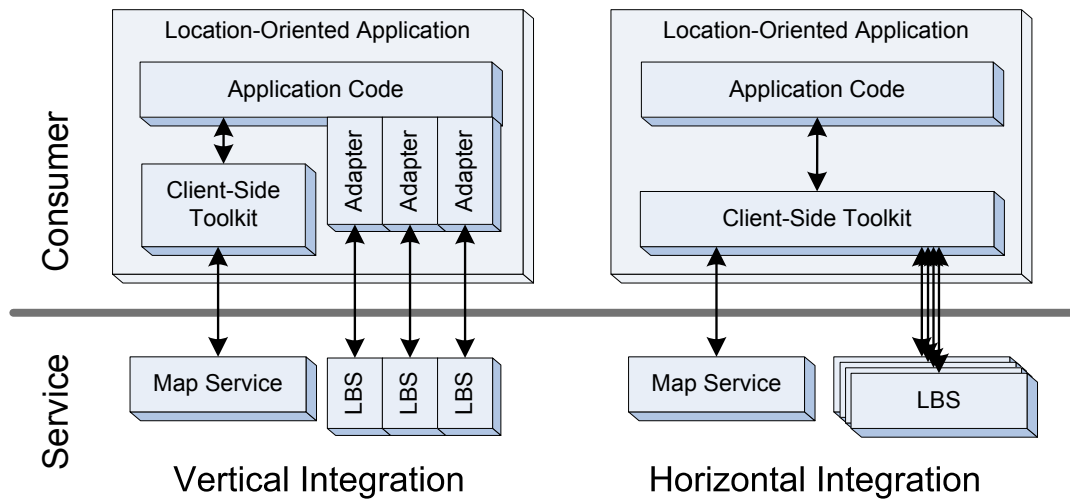


Figure 3: Horizontal vs. Vertical Integration

because the REST approach allows to use linked resources based on how they are interlinked [1]. Aggregation and filtering become tasks that work in the same way as generic feed-based toolsets,¹ but they could be enriched with location-specific functionality. In a very simple way, location-based access to feed already is possible with GeoRSS and its ability to encode spatial information about feed entries in the entry metadata, but since there is no query mechanism and no other way to access these geocoded feeds in a way that is structured according to location, GeoRSS does not support any access pattern other than passive consumption of time-ordered feeds.

Clients using our LBS have two options of combining LBS from various publishers: if they just want to provide a combined view, they can simply do so by using the main mechanisms of Tiled Feeds. They can even easily expose their combined view as a Tiled Feed, because aggregation and filtering mechanisms can be seen as rather basic tasks in a feed publication pipeline. However, clients may also have additional knowledge about the value-added services exposed by one or more of the Tiled Feeds they use, and they may decide to use this additional knowledge to provide additional services. In this case, the combination of the various LBS is more than just aggregation of basic Tiled Feeds, because the client uses additional knowledge to extract more semantics about resources, or to follow additional links.

Another possible scenario is that clients choose to expose their services as value-added LBS, in which case they would enrich the resources they are exposing (i.e., the Tiled Feeds) with additional metadata and/or links. The most important observation in all of these scenarios is that they work in exactly the same way as the human Web works: some minimal standards on how to expose data and how to interlink data, and then allowing an extensible ecosystem of additional resources, so that service providers as well as consumers can work in an environment that is simple, open, and extensible.

¹ *Yahoo Pipes* and the *Yahoo Query Language (YQL)* are two examples for toolsets that allow sophisticated tasks to be implemented based on feeds as data sources.

6. FUTURE WORK

Many things remain to be done. The Tiled Feeds architecture and protocol requires a complete formal specification, which we are in the process of drafting. The current Tiled Feeds server and clients are proof-of-concept prototypes, and are not particularly polished. We are working toward one-click installation of server software (so that we can expose a greater variety of LBS as Tiled Feeds) and more user-friendly clients.

Given the large number of LBS already deployed, it would also be useful to create server-level adapters for these sources of data — to allow open, standardized feed-based access to these information silos. This would serve to demonstrate the general utility of the Tiled Feeds models of LBS, the virtues of horizontal integration, and the advantages of standardized LBS access and interactions in general.

7. CONCLUSIONS

Location-based services nowadays often use a tightly coupled design where the services are tied to some specific frontend. We propose LBS that are rooted in Web architecture and REST, and expose spatial information in interlinked and query-capable resources. This design allows LBS to use a common architecture and makes it easier for clients to use and possibly combine LBS from a variety of service providers, which they can use to implement client-side customization and personalization. In the same way as feeds have become an essential “container mechanism” for providing access to news feeds and similar resources on the Web, we proposed that “Tiled Feeds” provide spatial access to location-oriented services. Because our proposal is based on REST, we claim that it is much easier to combine and even republish LBS that it is in more vertically integrated scenarios.

8. REFERENCES

- [1] ROSA ALARCÓN and ERIK WILDE. From RESTful Services to RDF: Connecting the Web and the Semantic Web. Technical Report 2010-041, School

of Information, UC Berkeley, Berkeley, California, June 2010.

- [2] ROSA ALARCÓN and ERIK WILDE. Linking Data from RESTful Services. In *Third Workshop on Linked Data on the Web*, Raleigh, North Carolina, April 2010.
- [3] ROY THOMAS FIELDING and RICHARD N. TAYLOR. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.
- [4] JOE GREGORIO and BILL DE HÓRA. The Atom Publishing Protocol. Internet RFC 5023, October 2007.
- [5] DOMINIQUE GUINARD, VLAD TRIFA, and ERIK WILDE. A Resource Oriented Architecture for the Web of Things. In *Second International Conference on the Internet of Things (IoT 2010)*, Tokyo, Japan, November 2010.
- [6] IAN JACOBS and NORMAN WALSH. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.
- [7] IRIS A. JUNGLAS and RICHARD T. WATSON. Location-Based Services. *Communications of the ACM*, 51(3):65–69, March 2008.
- [8] MARTIN KOFAHL and ERIK WILDE. Location Concepts for the Web. In IRWIN KING and RICARDO BAEZA-YATES, editors, *Weaving Services and People on the World Wide Web*, pages 147–168. Springer-Verlag, Heidelberg, Germany, August 2009.
- [9] YIMING LIU and ERIK WILDE. Scalable and Mashable Location-Oriented Web Services. In BOUALEM BENATALLAH, FABIO CASATI, GERTI KAPPEL, and GUSTAVO ROSSI, editors, *10th International Conference on Web Engineering (ICWE 2010)*, volume 6189 of *Lecture Notes in Computer Science*, pages 307–321, Vienna, Austria, July 2010. Springer-Verlag.
- [10] ALEXANDER MAYRHOFER and CHRISTIAN SPANRING. A Uniform Resource Identifier for Geographic Locations ('geo' URI). Internet RFC 5870, May 2010.
- [11] MARK NOTTINGHAM. Web Linking. Internet Draft draft-nottingham-http-link-header-10, May 2010.
- [12] MARK NOTTINGHAM and ROBERT SAYRE. The Atom Syndication Format. Internet RFC 4287, December 2005.
- [13] OPEN GEOSPATIAL CONSORTIUM. OGC Web Map Service Interface. OGC 03-109r1, Version 1.3.0, January 2004.
- [14] OPEN GEOSPATIAL CONSORTIUM. Web Feature Service Implementation Specification. OGC 04-094, Version 1.1.0, May 2005.
- [15] OPEN GEOSPATIAL CONSORTIUM. OGC KML. OGC 07-147r2, Version 2.2.0, April 2008.
- [16] OPEN SOURCE GEOSPATIAL FOUNDATION. Tile Map Service Specification. http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification, 2009.
- [17] CESARE PAUTASSO and ERIK WILDE. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In JUAN QUEMADA, GONZALO LEÓN, YOËLLE S. MAAREK, and WOLFGANG NEJDL, editors, *18th International World Wide Web Conference*, pages 911–920, Madrid, Spain, April 2009. ACM Press.
- [18] CESARE PAUTASSO, OLAF ZIMMERMANN, and FRANK LEYMAN. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In JINPENG HUAI, ROBIN CHEN, HSIAO-WUEN HON, YUNHAO LIU, WEI-YING MA, ANDREW TOMKINS, and XIAODONG ZHANG, editors, *17th International World Wide Web Conference*, pages 805–814, Beijing, China, April 2008. ACM Press.
- [19] ANDREI POPESCU. Geolocation API Specification. World Wide Web Consortium, Candidate Recommendation CR-geolocation-API-20100907, September 2010.
- [20] HANAN SAMET. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.
- [21] JOCHEN SCHILLER and AGNÈS VOISARD. *Location Based Services*, chapter 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
- [22] STEVE VINOSKI. Serendipitous Reuse. *IEEE Internet Computing*, 12(1):84–87, January 2008.
- [23] ERIK WILDE. Feeds as Query Result Serializations. Technical Report 2009-030, School of Information, UC Berkeley, Berkeley, California, April 2009.
- [24] ERIK WILDE and ALEXANDROS MARINOS. Feed Querying as a Proxy for Querying the Web. In *Eighth International Conference on Flexible Query Answering Systems*, volume 5822 of *Lecture Notes in Artificial Intelligence*, pages 663–674, Roskilde, Denmark, October 2009. Springer-Verlag.