# Scalable and Mashable Location-Oriented Web Services

Yiming Liu and Erik Wilde

School of Information
UC Berkeley

**Abstract.** Web-based access to services increasingly moves to location-oriented scenarios, with either the client being mobile and requesting relevant information for the current location, or with a mobile or stationary client accessing a service which provides access to location-based information. The Web currently has no specific support for this kind of service pattern, and many scenarios use proprietary solutions which result in vertical designs with little possibility to share and mix information across various services. This paper describes an architecture for providing access to location-oriented services which is based on the principles of *Representational State Transfer (REST)* and uses a tiling scheme to allow clients to uniformly access location-oriented services. Based on these *Tiled Feeds*, lightweight access to location-oriented services can be implemented in a uniform and scalable way, and by using feeds, established patterns of information aggregation, filtering, and republishing can be easily applied.

**Keywords:** Web Services, Location-Oriented Services, REST, Loose Coupling.

## 1 Introduction

The *mobile Web*, the mobile access to Web-based resources, has gained a lot of momentum and attention. The increasing availability of sophisticated devices for mobile Web access (smartphones and netbooks) means that an increasing share of Web users access the Web from mobile settings, and in many cases, these users are interested in localized services. Complementing this, the increasing usage of the mobile Web produces an increasing amount of localized data ("location trails" of users and networked objects), which allows novel and personalized access to services based on this localized data. In summary, location-orientation on the Web has seen a sharp incline in interest and sophistication, and it is likely that this development will continue for the foreseeable future.

An increasing share of the services provided over the Web, either in the form of Web pages for browser-based UIs, or in the form of Web service APIs for use by applications, use or support location in some form, but the Web itself still is a location-unaware information system [7], which means that in many cases, the services or information made available are hard to repurpose and reuse. A lot

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 307–321, 2010.

of location-oriented services nowadays use Web technologies, but in many cases, the spatial component of service design is service-specific, and thus cannot be easily reused or recombined across services. A first change in that landscape is the W3C's current work towards a geolocation API [19], which allows scripting code to access a client's location services, but this API is still in its draft stage, and only covers the case where location information should be made available on the client.

In the general area of *Geographic Information System (GIS)* research, Web-oriented access to GIS systems has seen only little advances beyond the traditional model of client-based access to a centralized GIS server. *Web GIS* [1] are often simply regarded as Web interfaces for GIS systems, which means that they replace other client-side technologies with browser-based access to a single GIS. The *Open Geospatial Consortium (OGC)* has released a number of standards for accessing GIS systems using Web services, but all of these specifications are based on RPC models of interaction, and thus do not apply principles of Web architecture to GIS systems.

In this paper, a more loosely coupled architecture [17] of accessing GIS services is described, which can be used to provide access to any kind of geospatial service. It is based on the principles of *Representational State Transfer (REST)*, and thus is designed around interlinked resource representations served through a uniform interface, instead of using the RPC model of a sophisticated method set provided through a single endpoint.

## 2 Related Work

The *Open Geospatial Consortium (OGC)* has defined the *Web Map Service (WMS)* [11] and *Web Feature Service (WFS)* [13] 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 map imagery and map features as Web resources, they expose functions that can be remotely called to request map imagery and features. The RESTful approach of our work, on the other hand, exposes information in the interlinked fashion of Web resources, and thus clients 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].

The idea of tiled access to map information in general 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 as Google Maps, Flickr do not use the tiling concept

to organize and deliver spatial information. Proprietary APIs for location-base services are typically designed in ad-hoc fashion, with different types of access capability. The simplest instantiations, such as Flickr, allow queries by point and radius, while some services may allow user-specified bounding boxes.

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 Feeds as RESTful Web Services

Starting from the principle of loose coupling [17], a popular and well-established method for implementing RESTful access to collections of resources is *Atom* [10], a language for representing feeds. Extending the idea of the feed as a single representation of a collection, the model of interaction with feeds can be extended to cover queries and other interaction mechanisms [24], so that feeds and interactions with feeds turn into a more feature-rich model of interaction with resource collections. This model, however, is still a read-only model, but the *Atom Publishing Protocol (AtomPub)* [5] can be used to extend it into a model that also allows write access to collections.

While feeds satisfy the criteria for RESTful resource access (URI-based access, self-describing representations, interlinked information resources), additional feed-based capabilities for filtering, querying, and sorting collections are not yet standardized. Some feed-based services such as Google's *GData* or Microsoft's *OData* introduce their own extensions to add this functionality to feed-based services, but none of these so far has reached critical mass to be accepted as a new standard. In this paper, we focus on describing an access model to spatial information that builds on feeds to define a lightweight interaction model with spatial data that provides access to spatial information services. This model lays the groundwork for establishing scalable and open interaction patterns with these services. We highlight some of the issues for a more flexible and customizable access to these services in the final parts of the paper, but leave these issues for further work.

## 4 The Tiled Feed Model

In this section, we present a REST-based model for delivering mashable location-oriented services, based on multiple potential data sources. The design goals for this model are to address four areas of concern in the design of location-oriented Web services:

**Remixability.** Much of the geospatial information on the Web remains standalone and service-specific. Application scenarios involving multiple sources of geospatial information — now becoming common with highly localized and personalized services — require ad-hoc integrations with each of the desired data sources in turn. One of our primary goals in this model is to facilitate the creation of novel location-oriented services that reuse and remix available geospatial data.

**Loose coupling.** Much as we seek to liberate geospatial data from the constraints of service-specificity and tight coupling, clients of our model should not be tightly bound a specific system of our own design.

**Scalability.** With the rapid growth of location-oriented services and users, a model that scales cheaply and efficiently, using existing technology and know-how is particularly desirable.

**Ease of Deployment.** A driving factor of the rapid growth of the Web rests with ease of development of Web sites via HTML. In a similar vein, many individuals and organizations may have geospatial data that would be valuable if made available as open Web services, and more valuable if easily remixed with data from other entities. The development of applications that use existing data and services should be made as easy as possible.

Our solution is called *Tiled Feeds*, geospatial data feeds based atop various geographic information systems. Tiled feeds can be published by an individual or organization with first-hand geospatial information, a commercial data provider, or even motivated third-parties, much like conventional RSS or Atom feeds. As with conventional news or blog feeds, client applications may use one or more tiled feeds to build, mix, and visualize spatial information as they wish.

### 4.1 Tiled Feed Architecture

Tiled feeds consist of, as the name implies, spatial tiles and data feeds. In this model, the world is partitioned into standard sets of tiles, at varying levels of resolution. Each tile is then published as an Atom feed with simple spatial extensions. Entries within the feed represent geospatial features that are located within that tile. The entries are represented in *Keyhole Markup Language (KML)* [15] or *Geographic Markup Language (GML)* [12], which are standard markup languages for representing geospatial information features, such as points, lines, polygons, etc.

**Tiles.** In the tiled feed system, tiles represent a standardized spatial unit for access to feeds. Tiling is a well-known technique for spatial division and aggregation of georeferenced data. Size of the tiles indicates level of resolution — larger tiles offer less detail but more coverage of an area, and vice versa for smaller tiles. Repeated tiling of the same area provides different levels of resolution, or "zoom levels", for a tiled map. For example, a map may be divided into four large tiles at the first level of detail, and 16 smaller tiles at the second level.

There are many possible tiling schemes, but we adopt the basic tiling approach used in most Web-based mapping services [21,4]. The base map is a map of the world, in WebMercator or EPSG:3857 projection [2]. We subdivide this base map into smaller and smaller tiles, using a conventional quadtree-based [20] tiling algorithm.

In our canonical tiling scheme for tiled feeds, the world map is recursively divided into four smaller tiles. The top-level tile is set to be the entire base map. This constitutes zoom level 0. We then divide the top-level tile into four equal

square tiles, creating the tiles for zoom level 1. Each of the four new tiles in level 1 can be divided again, in four, to create 16 tiles in level 2. This process continues for each new level of resolution desired. We recommend a maximum of 20 levels, the same provided by most mapping services. Due to the properties of the Mercator projection[1], we restrict the base map to latitudes between -85.05112878 and 85.05112878 — the same restrictions used in both Google Maps and Bing Maps. This also conveniently creates a square map, which yields square tiles simplifies tile-based distance computations.

For identification and linking purposes, an addressing value is assigned to each tile. Both the tiling and the addressing scheme are illustrated in Figure 1, with the world recursively divided and each area numbered 0 to 3. In essence, the top-left tile, covering the top-left sector of its parent tile, is always given a value of 0. The top-right tile is always given a value of 1, the bottom-left given 2, and the bottom-right is given 3. Thus, the *tile key*, a hash string that uniquely identifies a particular tile, is given by the concatenation of the addressing values of its parent tiles. For example: the key "0-1" identifies the top-right tile of the top-left tile of the base map. The length of its *tile key* represents the zoom level of the tile.

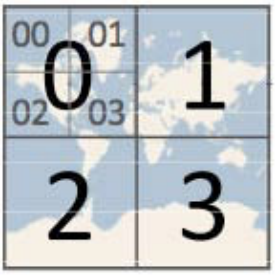


**Fig. 1.** Quadtree-based tiling and addressing scheme

**Tile Feeds.** Each tile is represented by an Atom feed, called the *tile feed*. The tile feed carries metadata information and descriptions similar to an ordinary Atom feed. The tile is referenced by a unique URI. For example, the tile representing the entire world would have an ID of `http://example.com/feed/top`. The northwestern tile at zoom level 1 would have an ID of `http://example.com/feed/0`. The tile feed links to neighboring, contained, and containing tiles, for RESTful navigation of the map.

However, the tile feed is extended with some simple spatial properties. In particular, a tile feed has the following specific properties:

- The `atom:id` element contains the URI for the tile.
- Four `atom:link` elements point to the neighboring tiles. Links with the relations of *north*, *south*, *east*, and *west* point to the URIs of neighboring tiles in the four cardinal directions. If there is no tile in that direction[2], the corresponding link element is omitted.
- An `atom:link` element points to the containing parent tile. A link with the relation of *up* points to the URI of the tile one zoom-level up, which contains the current tile.

[1] Singularities are present on both poles. The line at the top of a Mercator map, is in actuality a point — the north pole. The same applies for the line at the bottom of a Mercator map.

[2] For example, the tile 0 has no neighboring tile to its north.

– Four `atom:link` elements point to the tiles at the next zoom level, contained within the current tile. A link with the relation of *down-northwest* points to the URI of the tile at the northwestern quadrant, the *down-northeast* relation for the northeastern quadrant, *down-southwest* for the tile at southwestern quadrant, and *down-southeast* for the tile at the southeastern sector.

With these extensions, subscribing clients of the tile feed can navigate the world using the provided links, load neighboring tiles, and retrieve information as needed. Each individual `atom:entry` in the tile feed consist of a resource available within this tile, representing some GIS feature and its attributes, written in KML or GML. For read-only feeds, these entries are refreshed when its back-end data source is updated. For writeable feeds, AtomPub may be used as a standard means for posting, editing, and deleting resource items within the feeds, creating more dynamic tile feeds.

Features and attributes are provided by the underlying geographic information system. The system may be, for example, geographic information databases, third-party location-oriented services, or standardized GIS web services. We discuss the methods for integration with these data sources in more detail in Section 5.

A tile feed may support paging, and does so according to the standardized feed paging mechanism [8]. A specific page is requested with a *page* query parameter in its URI, with a value equals to some positive integer representing desired page number. In this case, additional `atom:link` elements will point to the URIs of the previous, next, first, and last pages, as directed in the feed paging specification. More advanced operations, such as filtering, sorting, or querying, may also be supported; we discuss this in more depth in Section 5.

**Implementation.** We have implemented proof-of-concept tiled feeds using freely available geospatial datasets and geospatial-aware relational databases. The popular PostgreSQL RDBMS can be augmented for GIS *geometry* columns and spatial query functions using the PostGIS[3] add-on. GIS features were loaded into PostgreSQL tables, and a thin service layer is written to serve GIS features in the tiled feed format as described. We created, as tiled feeds, the locations and addresses of 861 Amtrak[4] stations in the United States, the addresses and species of 64,318 street trees maintained or permitted by the city of San Francisco, and 25,928 magnitude 3.0+ earthquakes occurring in the United States in 2008.

An excerpt from the feed is provided in Listing 1. As noted in the previous section, the feed, in addition to resource entries and standard Atom metadata, also contains navigational extensions pointing to neighboring tile feeds, as well as the feeds of its parent tile and child tiles.

The Amtrak tiled feed effectively describes points of access for inter-city passenger rail service in the United States, and is useful as an example of an information collection covering a large area. The San Francisco trees collection, on the

[3] Available at `http://postgis.refractions.net/`

[4] Amtrak is the sole intercity passenger rail service in the United States.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns:fh="http://purl.org/syndication/history/1.0" xml:lang="en-US"
     xmlns="http://www.w3.org/2005/Atom">
  <id>http://tfserver/tiles/02301021</id>
  <link type="text/html" rel="alternate" href="http://tfserver/tiles
       /02301021"/>
  <link type="application/atom+xml" rel="self" href="http://tfserver/tiles
       /02301021"/>
  <title type="text">Tile 02301021</title>
  <updated>2009-11-04T11:10:46-08:00</updated>
  <author>
    <name>TileFeed Generator</name>
  </author>
  <link type="application/atom+xml" rel="http://tfserver/tiledfeeds/
       relation/north" href="http://tfserver/tiles/02301003"  />
  <link type="application/atom+xml" rel="http://tfserver/tiledfeeds/
       relation/south" href="http://tfserver/tiles/02301023"/>
...
  <link type="application/atom+xml" rel="http://tfserver/tiledfeeds/
       relation/down-southwest" href="http://tfserver/tiles/023010212"/>
  <link type="application/atom+xml" rel="http://tfserver/tiledfeeds/
       relation/down-southeast" href="http://tfserver/tiles/023010213"/>
  <fh:complete/>
  <entry>
    <id>http://tfserver/items/287</id>
    <link type="text/html" rel="alternate" href="http://tfserver/items/287"
         />
    <title>ACA</title>
    <updated>2009-11-04T11:10:44-08:00</updated>
    <content type="application/vnd.google-earth.kml+xml">
      <kml xmlns="http://www.opengis.net/kml/2.2">
        <Placemark>
          <name>ACA</name>
          <description>100 I Street, Antioch-Pittsburg, CA</description>
          <Point>
            <coordinates>-121.815132000194,38.0180849997628</coordinates>
          </Point>
        </Placemark>
      </kml>
    </content>
  </entry>
   ...
```

**Listing 1.** An example, unpaged tile feed containing Amtrak stations for the tile 02301021, a tile covering northern California

other hand, is a demonstration of a highly localized yet relatively dense dataset. Such collections are potentially useful for community-based location-oriented services, such the creation of a municipal dashboard for local government or a neighborhood environmental program.

We use a typical page-caching pattern for optimizing feed services. When a tile feed is requested, its feed is generated, and a static file containing the entire feed is saved to disk. As long as the underlying information is not updated, the file is not regenerated, and the feed service simply serves the static file for every subsequent request for the tile. The feed itself supports standard HTTP `ETag` based caching, for reader-level caching.

Our service implementation supports 20 zoom levels for the entire world, which yields $\sum_{i=1}^{20} 4^i$ tiles. However, for typical datasets, only a small number of tiles are in service. While Amtrak covers much of the continental United States,

its stations are distributed unequally and mostly in coastal or urban areas. A similar effect occurs for the earthquake feed, where tiles of seismically active areas in the western United States tend to have data. When a tile without data is requested, the requesting client is simply presented with a standard HTTP 404 response. The response, too, can be cached unless a new feature is created within that tile.

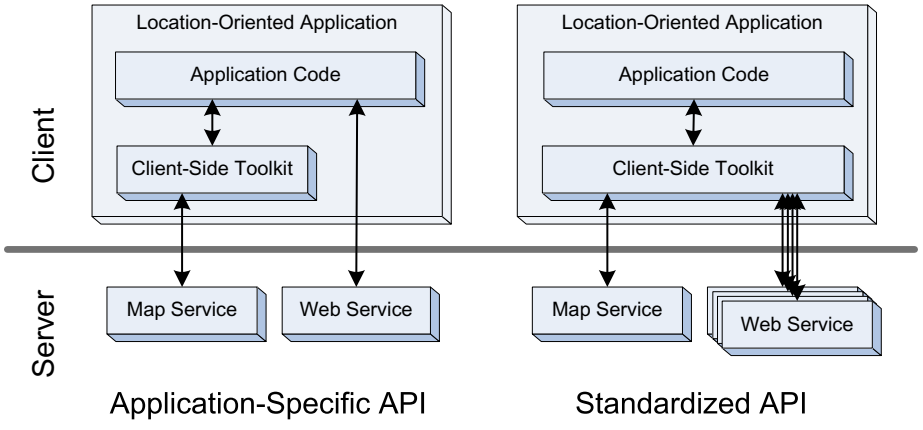


**Fig. 2.** Architectural comparison between a conventional location-aware service built upon tightly-coupled, application-specific APIs, and a service built upon tiled feeds

**Discussion.** Tile feeds are accessed by clients in the same manner as ordinary Atom feeds. Compared with current client-server architectures for location-oriented applications, the tiled feed model creates a standardized interface for interacting with location-oriented services. An architectural comparison diagram is provided in Figure 2.

The tiled feed design accomplishes the four design goals we initially laid out for access to geospatial information. For a typical location-oriented application, tiled feeds provide access to multiple resources in an easy, uniform way, providing for easy remixability. For example, currently, a mobile application for travelers may write code to interact with several APIs: one for flight updates and traffic information, another for restaurant reviews, etc. The application must also download map visualizations via a specific mapping service such as Google Earth, Bing Maps, or standard services like the *Web Map Service (WMS)* [11]. In contrast, with a tiled feed, the application can simply include a tiled feed reader and subscribe to the tile feeds for the areas needed. Remixing data — such as traffic conditions, restaurant reviews, and map visualization — is as simple as subscribing to two different tiled feeds for the same tile.

Feed accesses are loosely coupled, much as Atom feeds themselves are loosely coupled from their internal systems and representations. Interactions with the tiled feeds take place via standard HTTP requests to resources and resource collections. This also provides advantages in simple scalability and deployment, in that

existing techniques for scaling Web servers and feeds (as opposed to GIS-specific servers and optimizations) can be directly applied to scaling tiled feeds.

As an example, tile feeds can be cached as typical feeds. At client level, entity tag-based caching [3] for RSS and Atom-based feeds, can be used with no modification for a tile feed. Since tile feeds corresponds to unique URIs, conventional Web caches and caching proxies can be used for multi-level caching. At server level, simple page caching techniques can be used to generate and serve static feeds. This enables feed servers to satisfy a large number of requesting clients at very little additional expense. If no data is available for a particular tile — an expected outcome for many data sources covering only specific locations — simple HTTP 404 responses, or redirects to the closest available tile, can be used. It would then be up to the tiled feed client to handle error responses and decide whether to follow redirects to another tile.

The tiled feed model, as described, presents distinct advantages over existing methods for accessing and using geospatial information. As Atom feeds, the data presented is conveniently consumable via standard tools. Clients can choose the tiles that interest them, at any particular level of geographic resolution — thus avoiding the complexity and overhead of geospatial queries. Further, the tile model provides for simple horizontal scalability. By mediating queries for geospatial data into predictable, RESTful patterns, techniques for typical HTTP- and feed-based content delivery can be directly applied for these geospatial feeds as well. Perhaps most importantly, tiled feeds can be easily remixed and combined. Much as Web-based news and blogs are delivered via standard feeds, processed by a variety of reader software, and syndicated across many web sites, geospatial data in tiled feeds can also be consumed in this manner. Remixability also lowers the barrier to reusing data, and encourages the creation of information dashboards and mashups.

## 5 Publishing Tiled Feeds

A tiled feed publisher may publish data from many types of geospatial information sources. We describe three implementation scenarios of tiled feeds atop existing data sources, including geospatial databases, proprietary location-oriented services, and standard GIS services. A diagram illustrating the three particular scenarios is presented in Figure 3.

### 5.1 Geospatial Databases

Much geospatial data exist in GIS-specialized relational databases and static files. Both consumer and enterprise-grade relational databases now possess geospatial information storage and querying capabilities, implementing and extending the OpenGIS *Simple Feature Access* [14] to varying degrees. Static geospatial datasets, often provided as *shapefiles* of vector or raster data, can be imported to these databases using widely available tools, as we have done in our implementations.

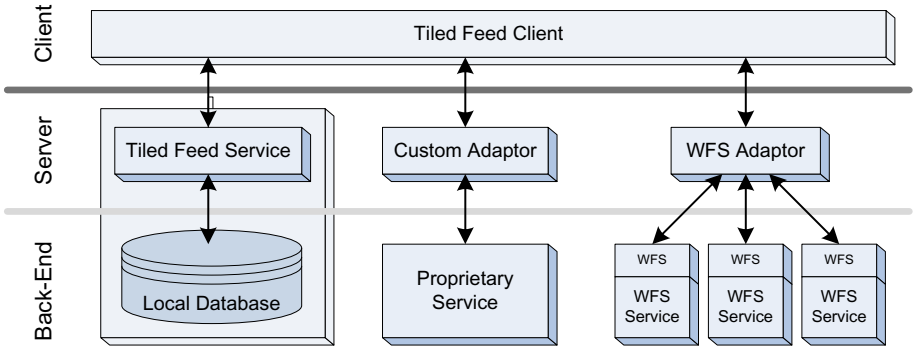


**Fig. 3.** Implementations of Tiled Feeds atop various geospatial data sources

A tiled feed implementation has the greatest potential capability and flexibility when directly integrated with such a spatial database, at the cost of a tight coupling. The feed generator would be able to create any tile feed on request, and provide grouping and filtering capabilities cheaply. For example, listing all features and associated attributes in a given tile is a trivial spatial SQL query.

Due to the tight coupling, the publisher of the tiled feed must have direct access to the database, and must create a tiled feed service based on the data. As tiled feed services are simply Atom feeds with spatial extensions, this should be relatively painless. GIS server software such as GeoServer[5], which integrate with geospatial databases to provide data access interfaces, may be also extended to publish tiled feeds, so as to reduce custom implementation work.

### 5.2 Proprietary Location-Oriented Services

Tiled feeds can also be implemented atop existing location-oriented services with APIs. The growth of location-aware devices has prompted many Web applications, such as Flickr or Twitter, to provide geotagged data. A tiled feed can be created from these proprietary APIs via a software adapter that queries the underlying API for information within the requested tile boundaries.

Adapted tiled feeds based on proprietary APIs may have widely varying levels of capability and flexibility, depending on the spatial queries offered by the underlying API. There is significant potential for mismatches in desired capability and offered capability. For example, the Flickr API offers radius-based search for photos; the query takes as input a geographic coordinate and a set of query terms. Obviously this query is not fully compatible with the square geometries of tiles and multiple zoom levels of the tiled feed model. Furthermore, advanced features such as filtering or paging entries, or writing new entries into the tiled feed, depend upon support from the underlying API.

[5] Available at `http://geoserver.org/`

There are potential fixes. Algorithmic approaches may be used to mask some spatial querying mismatches, such as using geometric approximations of the tile using a crafted radius-based query. Adapting existing sources of data also enables a wider variety of spatial data to become available as tiled feeds, and promotes data reuse and remixing. It is also more likely that third parties would be able to offer tiled feeds based on these APIs.

### 5.3 Standardized Services

Somewhere in the middle of the spectrum of potential capability are tiled feed implementations based on standard GIS web services. The *Web Feature Service (WFS)* [13], an *Open Geospatial Consortium (OGC)* standard, provides a standardized interface to geospatial data. The Web Feature Service defines a set of operations for querying, creation, update, and deletion of geographic features contained within a geospatial dataset. Results can be encoded in different formats, including the Geography Markup Language [12] and ordinary key-value pairs. The aforementioned GeoServer software is a reference implementation of the WFS, and many other GIS support WFS as well.

A tiled feed model can be implemented atop standardized GIS services such as WFS with relative ease, given the rich set of operations defined by these standards. Further, only one tiled feed adapter need be implemented, which then can be used to consume all WFS-compliant services. WFS-based tiled feeds may also support advanced features such as querying or sorting of tile entries, which are supported by the underlying API.

There still exists potential for capability mismatch between the tiled feed service and the underlying WFS or other standard API. Depending on the geospatial dataset, spatial features provided by WFS queries may be difficult to group into higher-level features. For example, a WFS service may return simple line geometries to establish the boundaries of a plot of land, which the tiled feed adapter must assemble into a polygon for its KML or GML resource entry. The WFS may also provide more advanced features than a tiled feed adapter requires, such as complex, multi-dimensional geometries that may be incompatible with a feed or feed reader. Such mismatches may be reconciled via additional metadata, or incompatible features may be excluded altogether from tiled feeds.

## 6 Experimental Client

To experiment with tiled feeds and assess the consumption of tiled feed-based services in both application development and practical use, we also implemented a prototype mobile tiled feed client using the iPhone SDK (shown in Figure 4).

The tiled feed client prototype has three views. In the first, the user adds tiled feeds as services to be consumed. The process is similar to adding feeds to a typical feed reader. Each feed can be hosted by a different organization, backed by a different geospatial dataset.

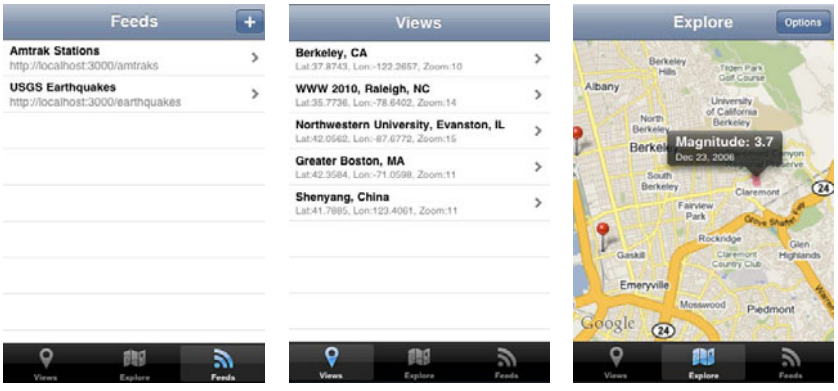


**Fig. 4.** A basic tiled feed reader. In (a), the user is consuming two location-based services as tiled feeds — locations of nearby Amtrak train stations, and recent M3+ earthquakes from the USGS, respectively. In (b), the user has subscribed to five locations of interest, including Berkeley, CA. In (c), the user selected the Berkeley location, is shown a mashup of Amtrak services and earthquakes at the location of Berkeley, CA.

Second, he identifies specific locations that interest him, such as "two blocks around UC Berkeley campus", or "Vienna, Austria". From this view of all "subscribed locations', he can immediately jump to a location on the map view. The tiled feed reader will retrieve features for that tile from all subscribed services and draw a mashup of the data in the map view. Alternatively, he can explore the world map manually, and the reader will load features for the current visible tile in the viewport from all services. As he navigates around the map, tiles are loaded and unloaded as needed — tiles with no features return 404 from the server, and are simply ignored by the client. At any particular location, the user may choose to add the current map view as a location of interest, adding it to the list of subscribed locations.

Views at specific locations may be personalized further. In the consumption of location-based services, users' information needs are highly context-sensitive. Depending on the usage, data useful at one location is not useful at another [6]. The tiled feed reader adapts to these user contexts; if a user is not interested in data from a particular service for a location, he can turn the service off for that location. For example, if a user is subscribed to a location-based service that provides traffic data, but is interested in the Los Angeles area solely for earthquake and air pollution information, he can turn off the (potentially data-heavy) traffic data for that location, but keep the earthquake and air pollution data on the map view.

This client development experiment demonstrates the significant advantages of the tiled feed model. From the user perspective, there is no central silo or any single point of failure — anyone with access to a geotagged dataset may publish it as a tiled feed. Creating mashups of geospatial services is easy, by simply using two or more services at a given location. From the developer perspective,

creating location-aware applications using tiled feed services required minimal code, mostly in the presentation layer to draw the maps, markers, and overlays. We used a standard Atom feed reader component, slightly extended to read the spatial properties from Tiled Feeds; no ad-hoc API adapters were required for adding additional services.

Both the server and the client caches retrieved tiles. Standard caching approaches yields significant benefit. The majority of accesses hits cache rather than triggering new geospatial queries.

In all, the tiled feed model makes information remixing from multiple data sources easy, and encourages the creation of in-context, personalized views and mashups of geospatial data.

## 7 Future work

With large amounts of geospatial data, tiles feed may contain tens of thousands of features in a given tile at average zoom level. Bandwidth consumption and server load increases significant at this scale, and tiled feed clients on constrained devices are often unable to process or visualize such amounts of data. As GIS datasets and wrapped geospatial APIs often contain large amounts of data, tiled feed servers and clients should be able to properly handle such tiles, if requested.

One solution involve clustering or aggregation on the server side. Aggregate features, identified by a special collection type that marks them as clustered versions of individual geospatial features. The tiled feed client is given the option of dereferencing aggregates into their component features.

Tiled feed filtering and sorting capability are also useful in this context. Feed querying in general remain an open problem [23], as the Atom standard does not adequately cover the query use case. There has been some prior work, including the aptly named *Feed Item Query Language (FIQL)* [9]. However, there are significant mismatches between FIQL and the set of operations desirable in a tiled feed query language. FIQL is largely interested in enabling the retrieval of markup elements in a feed, in a relatively domain-agnostic way. In contrast, tiled feeds may require queries against specific geospatial features contained within a feed, which would require some understanding of GML or KML datatypes.

It is also desirable for clients to autodiscover the query capabilities supported by a particular tile feed (otherwise, human intervention or configuration would be required to read new tiled feeds). For example, certain feed servers may not be able to execute complex queries like `GROUP BY` aggregation, or it may not be able to sort against a particular key. There is no appropriate mechanism to publish this information in FIQL. The design of an appropriately powerful query language for feeds in general, and tiled feeds in particular, is left for future work.

We are also working on more extensive user evaluations of tiled feed client programming (beyond our current simple client) and tiled feed-based applications, as well as quantitative scalability measurements of tiled feed server systems, to explore the advantages and drawbacks of the tiled feed model.

## 8 Conclusions

We have presented an architecture for providing uniform, lightweight access to location-oriented services, using feeds and geospatial tiling. Based on the principles of *Representational State Transfer (REST)*, tiled feeds is designed around interlinked resource representations served through a uniform interface, allowing loosely coupled and scalable access patterns. Tiled feeds can be easily created from existing sources of data and consumed by standard clients, reducing application complexity and easing application development. Established patterns of information aggregation, filtering, and republishing on the Web can be applied to tiled feeds, allowing the creation of novel information dashboards and mashups using geospatial data.

## References

1. Di Martino, S., Ferrucci, F., Paolino, L., Sebillo, M., Tortora, G., Vitiello, G., Avagliano, G.: Towards the Automatic Generation of Web GIS. In: Samet, H., Shahabi, C., Schneider, M. (eds.) 15th ACM International Symposium on Geographic Information Systems, pp. 57–64. ACM Press, Seattle (November 2007)
2. EPSG: 3857, WGS84 (2009), `http://www.epsg-registry.org/report.htm?type=selection\&entity=urn:ogc:def:crs:EPSG::3857\&reportDetail=short\&style=urn:uuid:report-style:default-with-code\&style_name=OGP%20Default%20With%20Code\&title=EPSG:3857`
3. Fielding, R.T., Gettys, J., Mogul, J.C., Frystyk Nielsen, H., Masinter, L., Leach, P.J., Berners-Lee, T.: Hypertext Transfer Protocol | HTTP/1.1. Internet RFC 2616 (June 1999)
4. Google: Map Overlays (2009), `http://code.google.com/apis/maps/documentation/overlays.html#Google_Maps_Coordinates`
5. Gregorio, J., de Hóra, B.: The Atom Publishing Protocol. Internet RFC 5023 (October 2007)
6. Kaasinen, E.: User Needs for Location-Aware Mobile Services. Personal and Ubiquitous Computing 7(1), 70–79 (2003)
7. Kofahl, M., Wilde, E.: Location Concepts for the Web. In: King, I., Baeza-Yates, R. (eds.) Weaving Services and People on the World Wide Web, pp. 147–168. Springer, Heidelberg (2009)
8. Nottingham, M.: Feed Paging and Archiving. Internet Draft draft-nottingham-atompub-feed-history-11 (June 2007)
9. Nottingham, M.: FIQL: The Feed Item Query Language. Internet Draft draft-nottingham-atompub-fiql-00 (December 2007)
10. Nottingham, M., Sayre, R.: The Atom Syndication Format. Internet RFC 4287 (December 2005)
11. Open Geospatial Consortium: OGC Web Map Service Interface. OGC 03-109r1, Version 1.3.0 (January 2004)
12. Open Geospatial Consortium: OpenGIS Geography Markup Language (GML) Encoding Specification. OGC 03-105r1, Version 3.1.1 (February 2004)
13. Open Geospatial Consortium: Web Feature Service Implementation Specification OGC 04-094, Version 1.1.0 (May 2005)

14. Open Geospatial Consortium: OGC Simple Feature Access. OGC 06-103r3, Version 1.2.0 (October 2006)
15. Open Geospatial Consortium: OGC KML. OGC 07-147r2, Version 2.2.0 (April 2008)
16. Open Source Geospatial Foundation: Tile Map Service Specification (2009), http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification
17. Pautasso, C., Wilde, E.: Why is theWeb Loosely Coupled? A Multi-Faceted Metric for Service Design. In: Quemada, J., León, G., Maarek, Y.S., Nejdl, W. (eds.) 18th International World Wide Web Conference, pp. 911–920. ACM Press, Madrid (April 2009)
18. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In: Huai, J., Chen, R., Hon, H.W., Liu, Y., Ma, W.Y., Tomkins, A., Zhang, X. (eds.) 17th International World WideWeb Conference, pp. 805–814. ACM Press, New York (April 2008)
19. Popescu, A.: Geolocation API Specification. World Wide Web Consortium, Working Draft WD-geolocation-API-20090707 (July 2009)
20. Samet, H.: The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys 16(2), 187–260 (1984)
21. Schwartz, J.: Bing Maps Title System (2009), http://msdn.microsoft.com/en-us/library/bb259689.aspx
22. Vinoski, S.: Serendipitous Reuse. IEEE Internet Computing 12(1), 84–87 (2008)
23. Wilde, E.: Feeds as Query Result Serializations. Tech. Rep. 2009-030, School of Information, UC Berkeley, Berkeley, California (April 2009)
24. Wilde, E., Marinos, A.: Feed Querying as a Proxy for Querying theWeb. In: Andreasen, T., Bulskov, H. (eds.) FQAS 2009. LNCS, vol. 5822, pp. 663–674. Springer, Heidelberg (2009)