ONLINE MAPS AND CLOUD-SUPPORTED LOCATION-BASED SERVICES ACROSS A MANIFOLD OF DEVICES

Michael Kröpfl [a, *], Daniel Buchmüller [a], Franz Leberl [b]

[a] Microsoft Corporation, One Microsoft Way, Redmond, WA, 98052 [USA] - (mkroepfl, danielbu)@microsoft.com
[b] Graz University of Technology, Institute for Computer Graphics and Vision, A-8010 Graz [Austria] - leberl@icg.tugraz.at

Commission IV, WG IV/5

KEYWORDS: Internet/Web, Mapping, Mobile, GPS/INS, LBS, Web Services, GIS, Databases, Georeferencing, Imagery, Aerial, Cloud Computing, Matching

ABSTRACT:

Online mapping, miniaturization of computing devices, the "cloud", Global Navigation Satellite System (GNSS) and cell tower triangulation all coalesce into an entirely novel infrastructure for numerous innovative map applications. This impacts the planning of human activities, navigating and tracking these activities as they occur, and finally documenting their outcome for either a single user or a network of connected users in a larger context.
In this paper, we provide an example of a simple geospatial application making use of this model, which we will use to explain the basic steps necessary to deploy an application involving a web service hosting geospatial information and a client software consuming the web service through an API.
The application allows an insurance claim specialist to add claims to a cloud-based database including a claim location. A field agent then uses a smartphone application to query the database by proximity, and heads out to capture photographs as supporting documentation for the claim. Once the photos have been uploaded to the web service, a second web service for image matching is called in order to try and match the current photograph to previously submitted assets. Image matching is used as a pre-verification step to determine whether the coverage of the respective object is sufficient for the claim specialist to process the claim.
The development of the application was based on Microsoft's® Bing Maps™, Windows Phone™, Silverlight™, Windows Azure™ and Visual Studio™, and was completed in approximately 30 labour hours split among two developers.

1. INTRODUCTION

1.1 Online Mapping and Location-Based Services (LBS)

Online mapping may be no older than two ISPRS-congress periods since 2004. Considering the evolution of Bing Maps or Google Maps, these data and user systems do continuously grow. They are essentially global with no consideration of national borders. They must be kept up-to-date. Further they combine vector-based geographic information with aerial and satellite imagery, streetside, indoor photography from industrial and community photo collections, and a plethora of metadata. Online maps have left the realms of desktop and notebook computers and have come to mobile devices. Be they smartphones, media players, navigation systems, tablet computers, slates or pads, mobile devices provide an excellent platform for LBS.

1.2 Mobile Devices and Cloud Computing

Location-based information is of great interest to the mobile user. Therefore it is very relevant that mobile computing is (a) increasingly miniaturized, (b) able to determine a position and attitude by Global Positioning Unit (GPS), Inertial Measurement Unit (IMU) and cell tower triangulation, (c) continuously connected to the Internet via cellular telephony or WIFI, (d) accessing very large databases and computing resources via the advent of cloud computing. The latter is now being offered by multiple providers such as Amazon, Microsoft®, and VMware. Additionally (e) the emerging capability of mixing computer resident and visualized data with natural scenes in an augmented reality (AR) context.

We can "bing" (or "google") shops or restaurants, the shortest path to a party, a picture note about an interesting sculpture, and an update of current road conditions. The web service is hosted in the "Cloud" and an "App" resides on the user's mobile device. The connection is provided by cell phone and/or WIFI technology. However, multiple devices come into play when planning an activity in an office or home on one set of perhaps stationary devices, when one or multiple users proceed to navigate towards a mobile activity on mobile devices, and when results need to get recorded and communicated on possibly a third handset. Such a scenario will expect that a map gets uploaded to the cloud for mobile availability for navigation to an action and for its documentation.

It is becoming increasingly important to develop innovative tools for users of location on mobile devices. Often these may address a simple geospatial application, but across multiple mobile and desktop devices and with the use of a common cloud-based backend.

In this paper, we provide an example of a simple geospatial application making use of this model, which we will use to explain the basic steps necessary to deploy an application involving a web service hosting geospatial information and a client software consuming the web service through an Application Programming Interface (API).

* Corresponding author.

In the development of the application Microsoft's® Bing Maps™, Windows Phone™, Silverlight™, Windows Azure™ and Visual Studio™ were used. All programming for both the client as well as the server components were completed in about 30 labour hours split among two software developers. In addition to the short development time, further advantages of using a cloud based web service rather than an in-house server are the minimal to non-existent investment cost and maintenance required, and the high availability of the service. Most cloud computing systems such as Windows Azure™, Amazon EC2™ and Google Apps™ offer free trials. This also helps to reduce the development costs for new applications.

While we implemented the client software only for the Windows Phone ecosystem, the web service could be consumed by multiple mobile operating systems including Apple iOS™, Google Android™ and BlackberryOS™.

1.3 Sample Application – "ClaimSnap" - Damage Investigation for Insurance Claims

As an example of an application following the above model, we present a simple workflow for investigation and documentation of home insurance claims – called "ClaimSnap". This involves a planning step in the insurance back office, a data collection step by one or more field agents, and an optional review of the collected data in the office, as visualized in Figure 1.

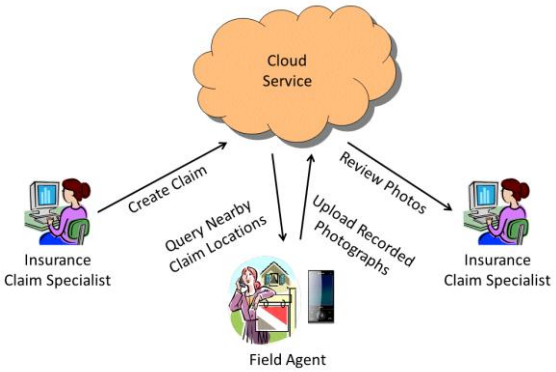

Figure 1 Example Workflow for Planning and Execution of Home Insurance Damage Investigation

The planning step is performed by an insurance claim specialist on a desktop computer and results in the creation of an insurance claim file including an address and details of the photography to be collected by the field agent. This claim is then uploaded to a cloud service, through which it can be discovered by a field agent's query for open claims based on location and other criteria. The query results are presented to the agent on a map which also allows navigation to the desired location. The field agent hence heads out to the given location, records the requested photographs covering details of the damage, and uploads the recorded data to the web service, where the claim is removed from the list of open claims. To confirm whether the uploaded images represent the scene well enough e.g. to allow later 3D reconstruction, an image matching service such as the one described in Kroepfl et. al. 2009 or the Bing Vision Service, can be consumed in order to verify that a sufficient number of matching photographs have been captured.

As the last step, the insurance claim specialist in the office can review the collected information, and decide to either request further data or close the file. This step can optionally be supported by feeding the images acquired through the ClaimSnap application to a service called Photosynth™, which computes a 3D reconstruction of the captured scene allowing a more immersive viewing experience for the claim specialist.
For simplicity of the application design, and to avoid distraction from the key components of a geospatial application, we decided to omit the notion of user identity, although this could easily be achieved by making use of the Windows Azure Access Control Service (ACS) (See References).

2. SYSTEM COMPONENTS

In this section we provide an overview of the system components, including the service and database used to store the data assets, the client application allowing a mobile user to access and contribute to the data assets, as well as the image matching service used to verify the completeness of the data acquired.

As mentioned above, several competing ecosystems are available for cloud computing, mobile mapping platforms (Amazon EC2™, Google Apps™), as well as mobile phone operating systems. For the specific implementation described in this paper, we used Windows Azure™ as the cloud operating system, in combination with SQL Azure™, a cloud-based version of Microsoft's® SQL Server™.

For the client software development we used the Silverlight™ functionality provided through the Windows Phone™ SDK v7.1. Silverlight™ is a development platform which, in combination with the .NET framework allows a software developer to design and program a visually compelling and easy to use graphical user interface within a very short development time. We chose C# as the programming language for the server and the client code. The .NET framework also support other languages such as C++ and Visual Basic.NET.

2.1 Service and Database

The core of the system is the ClaimSnap Service. Figure 2 illustrates the interactions of the service and all attached components. They break down as follows:

1. Receive requests from the client devices
2. Delegate storage requests to the database
3. Consume the Bing Vision Matching Service
4. Employ business logic to decide whether claims meet the completeness criteria
5. Interact with a web-based management portal to create claims and oversee the claims process. This portal is beyond the scope of this paper.

2.1.1 Service: The service exposes all methods that are consumed by the client application using Windows Communication Foundations (WCF). This allows for a clean separation between the describing data contract, binding, and endpoint configurations. The service itself is deployed to Windows Azure™ as a web node. Windows Azure™ allows for configuration-based scaling, redundancy, and geo-distribution of web and compute nodes.

As mentioned initially, the service forms the core of the system and acts as an accumulator, delegator, and processor of requests and data.

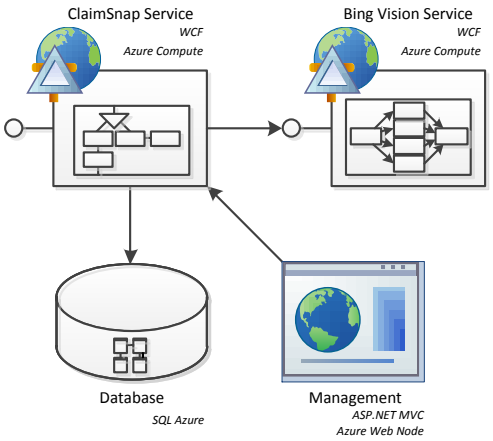

Figure 2 ClaimSnap Service and connected components

2.1.2 Database: The service sends storage requests (i.e. for claims and claim photo asset metadata) to a dedicated SQL Azure™ deployment. Again, the database server is not an on-premise installation that requires management by IT personnel but a virtual instance in the cloud, distributed over several machines for redundancy, and fully managed by Windows Azure™. As the data requirements grow, a virtual database instance can be scaled simply by changing its configuration through the Windows Azure™ Management Portal. In an on-premise solution scaling up could mean buying new hardware and extensive management overhead.

It's interesting to note that SQL Azure™ behaves no different than a traditional on-premise database when it comes to interfacing with it. In this specific solution the database holds the entity relationship model that describes how claims interact with their photo assets and how they relate and match against each other. Also the claim status is a modelled entity enabling client queries for claims in different processing stages. Figure 3 exhibits the relationship model in detail.

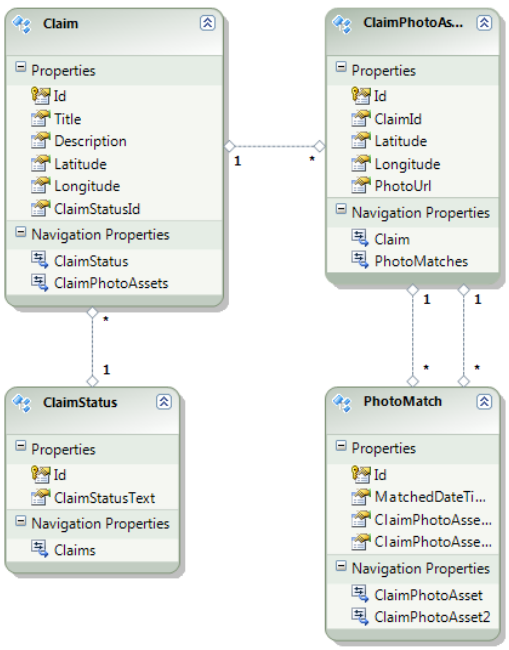

Figure 3 Entity Relationship Model

2.1.3 Bing™ Vision Service: The Bing Vision service exposes state-of-the-art image matching technology as a service. This technology is used for the visual product search features in both Windows Phone™ and the iPhone™ Bing™ app.

The matching technology is general enough enabling the service for matching photographs that have only a slight overlap. While this technology is traditionally used to find similar images or images of the same subject (i.e. a book, a landmark, etc.) the ClaimSnap service consumes the Bing™ Vision Service to compare an already submitted photograph for a claim. This introduces the central quality control component in the ClaimSnap system. The Bing™ Vision Service exposes its functionality through a (Representational State Transfer) RESTful interface, based on HTTP multipart/form-data requests and JSON-formatted responses for ease-of-use. The implemented metric will be described in detail in the next section.

2.1.4 Business Logic Component: As a central piece to the claims process, the business logic layer serves as the quality assurance component. The implemented logic demands that for each claim at least three photographs have to be recorded and that all three photographs have to match against each other to guarantee appropriate damage situation coverage; and allow for potential 3D reconstruction based on the photographs.

The business logic layer is consumed by the service through which it signals to the client application whether more photographs are required before a claim can be marked as complete. This component is located in the ClaimSnap Service deployment illustrated in Figure 2.

2.2 Windows Phone™ Client

Silverlight™ and the Windows Phone™ SDK offer a large variety of controls which can be used to design a versatile user interface leveraging the various input and output modes available on a state-of-the-art mobile device.

While it would exceed the scope of this work to describe the different controls in detail (a comprehensive documentation can be found in the Windows Phone™ Developer Resources and Quickstarts listed in the references), it is worthwhile to point out how little effort is required to implement a few very basic but common scenarios for LBS.

2.2.1 Hello World Map App for Windows Phone™: The first scenario explained is to display a map control and visualize some data points as pushpins superimposed on the map. After creating a new Windows Phone™ application in Visual Studio™ 2010 using a standard template, only three steps are required to achieve the desired functionality.

a) Create application and add map control
b) Add map layer
c) Draw pushpins on layer

First, a map control needs to be added to the design view window which provides a What-You-See-Is-What-You-Get (WYSIWYG) preview of the phone screen. This can be achieved literally with a few mouse clicks by selecting the "Map" control in the toolbox window and drawing a rectangular map region in the design view, such as presented in Figure 4.

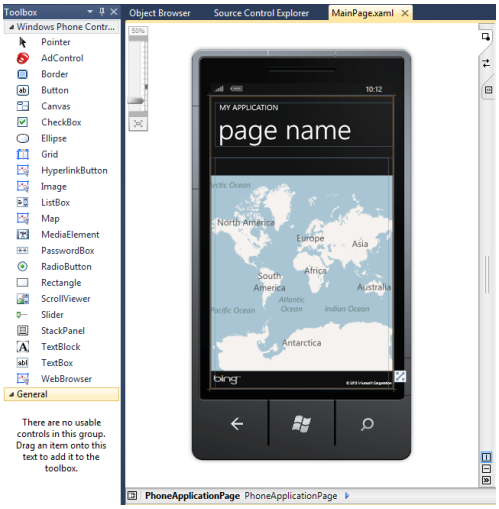

Figure 4 Toolbox and Design View for Windows Phone 7 Application Design using Silverlight™

This application could theoretically be built as is and deployed to a Windows Phone device. It would provide a simple map view which can be scrolled and zoomed using touch commands.

```
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle"
                   Text="MY APPLICATION"
                   Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle"
                   Text="page name"
                   Margin="9,-7,0,0"
                   Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>

    <my:Map
        Grid.Row="1"
        Height="610"
        HorizontalAlignment="Left"
        Name="map1"
        VerticalAlignment="Top"
        Width="480" />
</Grid>
```

Figure 5 Original XAML Code Snippet of Design Shown in Figure 4

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle"
               Text="Hello Map-World!"
               Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle"
               Text="Simple Map App"
               Margin="9,-7,0,0"
               Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>

<my:Map
    Grid.Row="1"
    Height="610"
    HorizontalAlignment="Left"
    Name="map1"
    VerticalAlignment="Top"
    Width="480">
    <my:MapLayer Name="mapLayer"/>
</my:Map>
</Grid>
```

Figure 6 Modified XAML Code with Added Map Layer and Updated Names for Page and Application

In order to be able to draw pushpins on the map, it is necessary to add a map layer, which serves as a canvas for showing geo-located information. The necessary code changes to the XAML file describing the elements of the UI are shown in Figure 5.

```
namespace HelloMapWorld
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();

            GeoCoordinate gc = new GeoCoordinate( // Melbourne Convention Center
                -37.82109049884058, // Latitude
                144.9548363685608); // Longitude

            map1.Center = gc;

            map1.ZoomLevel = 17.0;

            Random random = new Random();

            for (int i = 0; i < 10; i++)
            {
                Pushpin pushpin = new Pushpin();
                pushpin.Location = new GeoCoordinate
                    (gc.Latitude + (random.NextDouble() - 0.5) * 0.005,
                    gc.Longitude + (random.NextDouble() - 0.5) * 0.005);

                mapLayer.Children.Add(pushpin);
            }
        }
    }
}
```

Figure 7 Source Code to Draw Pushpins on Map Layer

In addition to adding the MapLayer, the page and application title on the page have also been updated. Finally, the actual source code to draw the pushpins has to be added; see Figure 7. The map is centred and zoomed to a certain viewport, and a number of randomly positioned pushpins are created on the previously created map-layer, as can be seen in Figure 8).

Figure 8 Completed Hello-World Application

Other, similarly useful tasks which we used during the creation of the ClaimSnap client software are the control of the built-in camera, as well as the location services APIs to access the current location through GPS and cell tower triangulation of the device. These tasks can be achieved using the Windows Phone™ SDK with a few lines of code, as demonstrated in Figure 9 and Figure 10.

Both examples make use of certain supporter classes (CameraCaptureTask or GeoCoordinateWatcher) to access the phone's functionality. Since the respective functions are

implemented asynchronously on the phone, a callback is required to define what should happen when the task completes.

```
BitmapImage _image;

public void CaptureImage()
{
    CameraCaptureTask cameraCaptureTask =
        new CameraCaptureTask();

    _image = null;

    cameraCaptureTask.Completed += (sender, e) =>
        {
            if (e.TaskResult == TaskResult.OK)
            {
                _image = new BitmapImage();
                _image.SetSource(e.ChosenPhoto);
            }
        };

    cameraCaptureTask.Show();
}
```

Figure 9 Code snippets to control camera

```
GeoCoordinate _location;

public void GetLocation()
{
    GeoCoordinateWatcher watcher =
        new GeoCoordinateWatcher(GeoPositionAccuracy.High);

    _location = null;

    watcher.StatusChanged += (sender, e) =>
        {
            if (e.Status == GeoPositionStatus.Ready)
            {
                _location = watcher.Position.Location;
            }
        };

    watcher.Start();
}
```

Figure 10 Code snippet to obtain geographic coordinates from the location API

2.2.2 ClaimSnap Client Design: When designing the ClaimSnap client app, we had primarily two user scenarios in mind, both assuming the user to be an insurance field agent:

- Query the web service for nearby claims and display them on the map; including the status of each claim
- Explore the details of claims, and allow capturing and uploading of supporting photographs to web service.

Therefore, we have organized the app into two pages according to these two user scenarios.

The first page as shown in Figure 11 a) contains a map showing the current location of the user (crosshairs-shaped pushpin) as well as the locations of nearby insurance claims. The map also contains buttons allowing zooming in and out. Below the map is a list of nearby claims, their status (New, SurveyStarted, SurveyCompleted, and Closed) and how many photographic assets have already been added. The list was implemented by using a Silverlight™ ScrollViewer as well as a StackPanel, into which TextBlock UI-Elements can be stacked.
Each of the claims can be selected by either touching the pushpin or the corresponding claim title in the list. This will cause the map to centre on the location of the selected claim, and highlight the respective claim in the list.

Below the ScrollViewer is a TextBlock showing the current app status. At the bottom of the screen are three buttons. The left button can be used to update the current position of the user using the location services of the phone. The centre button is used to initiate a query to the web service for nearby claims. The third button is only accessible if a claim is selected; to move to the second page as shown in Figure 11 b) and to manage the details of a claim.

The second page of the app is intended to show more details of a claim and is organized in four regions. The top-left part contains metadata about the claim such as title, description, and latitude / longitude of the object that the claim refers to.

a) b)

Figure 11 Graphical User Interface (GUI) for Windows Phone™ Client a) ClaimMap page showing an overview of different claims; b) Claim Info page showing details for a specific claim

Below that is an area containing one of the photographs of the claim object that have already been captured by an insurance field agent. The right side of the page holds a scrollable list of thumbnails of all related claim photographs. The colour of the thumbnail borders indicate whether the image has already been uploaded to the service and whether it has been accepted (green). Yellow means that it has not yet been uploaded, and red that it was uploaded but could not yet be matched to other photographs.

The three buttons below can be used to capture a new photograph, upload all newly captured photos or mark the active claim as "SurveyCompleted" (Left to right). The latter is only available if at least 3 photographs have been uploaded and could be matched to each other using the match service. Uploading image data consists of writing the file to Windows Azure™ Blob Storage in the cloud, and adding the new photo asset to the database. For the latter, as well as changing the status of a claim, the respective service endpoints of the ClaimSnap service are used.

3. RESULTS

After developing the ClaimSnap service as well as the client software, we have tested the system on various claim datasets we had previously created on the server database. Given the relatively short development time, both the cloud service as well as the client application were working well, and apart from network latencies on the 3G network, no major problems have been noticed. The data assed creation happened manually by using Microsoft® SQL Server™ Management Studio.

Figure 12 a) shows a graph of all 6 captured and matched images of a specific claim superimposed on the map, which was a feature we added late in the development to allow easier visualization of the geographic distribution of claim photo assets.

a) b)

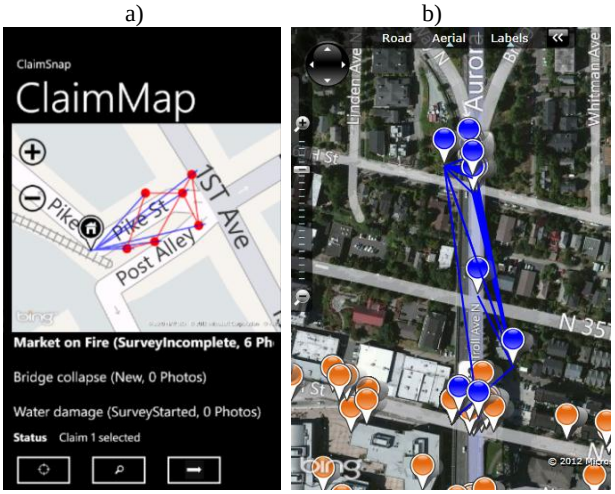

Figure 12 a) Graph of Matching Photographs on Mobile Client and b) on Desktop Silverlight™ Client

Figure 12 b) shows a similar match graph shown in a Desktop PC browser window, which is also using Silverlight™ technology for map and data visualization. The match graph was generated by the Bing Vision Service.

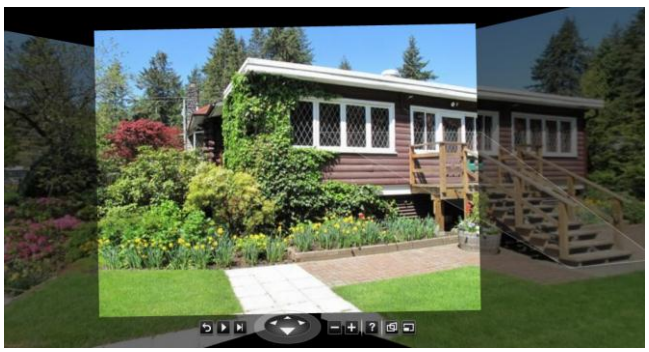

Figure 13 PhotoSynth™ 3D Reconstruction of a Building

In order to allow easier interpretation of the captured scene, the recorded photographs can further be processed using the Photosynth™ service, to form a 3D representation of the area. An example building shown in the Photosynth™ 3D image viewer is presented in Figure 13. Additionally, the reconstructed 3D scene can be observed in a top-down view (Figure 14), comprising a better and more accurate representation of the layout and the capture locations of individual photos than from just the GPS locations shown in Figure 12.

Concerning the application for insurance claims, we realized that it would have been advantageous to add a user identity model to the application, in order to avoid parallel processing of the same claim by multiple agents. Additionally, a notification function for new claims added to the database using push notifications to the clients would be a useful addition.

4. CONCLUSION

In this work, we have demonstrated that by using state-of-the-art technologies for cloud-based computing and databases (Windows Azure™ and SQL Azure™), as well as a modern smart phone architecture (Windows Phone™), a simple geospatial application can be developed within a very short time period of approximately 30 labour hours.

Figure 14 Top-Down View

This was possible by using high a high level programming language such as C# in combination with the Silverlight™ platform to create a visually rich and easy to use user interface involving natural user interface elements such as multi-touch panning and zooming of a map. Running the service and deploying an application to a set of mobile phones virtually free of cost, and available to any developer.

5. REFERENCES

Kroepfl, M., Wexler, Y., Ofek, E.
2010. Efficiently Locating Photographs in Many Panoramas. In: *GIS '10 Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*.

Microsoft® Corp. 2011, Read / Write / World 2011. Presented at *Where 2.0 Conference. 2011* http://www.readwriteworld.net Apr. 15. 2011 http://whereconf.com/where2011/public/schedule/detail/17794

Microsoft® Corp. 2012, Microsoft Photosynth™ http://www.photosynth.net/

Microsoft® Corp. 2012, .NET and Silverlight™ http://www.microsoft.com/net/ http://www.silverlight.net/

Microsoft® Corp. 2012, Windows Azure™ and Windows Azure™ Access Control Service http://www.windowsazure.com http://www.windowsazure.com/en-us/home/tour/access-control/

Microsoft® Corp. 2012, Windows Phone™ Development Resources and Quickstarts http://create.msdn.com/en-US/education/quickstarts http://create.msdn.com/en-US/education/basics/developer_resources

Apple Inc. 2012, Apple iOS™ http://www.apple.com/ios/

Amazon Inc. 2012, Amazon Elastic Compute Cloud™ (EC2) http://aws.amazon.com/ec2/

Google Inc. 2012, Google Android™ and Google Apps™ http://www.android.com/ http://www.google.com/enterprise/apps/business/