The Geobashing Architecture for Location-Based Mobile Massive Multiplayer Online Games

Bernhard Dieber, Thomas Grassauer, Jakob Mayring, Bernhard Rinner
Institute of Networked and Embedded Systems
Klagenfurt University
Klagenfurt, Austria
{*Bernhard.Dieber, Bernhard.Rinner*}*@uni-klu.ac.at*
{*Thomas.Grassauer, Jakob.Mayring*}*@edu.uni-klu.ac.at*

***Abstract*—The current trend towards more mobility will increasingly affect the gaming market over the next years. Location-based mobile massive multiplayer online games (MMMOG) will be a new paradigm in the gaming industry. Since building a game for a massive amount of mobile players is a challenging task, we have created a reusable architecture for location-aware mobile massive multiplayer online games. In this paper we describe our approach and present the Geobashing game, a location-based MMMOG implemented using our architecture. We evaluated our architecture to show its applicability.**

***Keywords*-Mobile gaming, Mobile Mixed Reality, Client–Server based services**

I. Introduction

Applications for mobile devices have become a fast growing market segment. The increasing number of smartphones which are GPS-enabled and have high-bandwidth internet connection is a driving factor for the development of location-based multi-user applications. Already today, massive multiplayer online games (MMOG) are very successful. Combined with the trend towards more mobility we expect that *mobile* massive multiplayer online games (MMMOG) will be an important part of the mobile application market in the coming years. Although the individual application scenario may be different, many of these applications will share some common properties. They are typically *client-server based*, *location-based* and enable *user interaction*. In this paper we present the Geobashing architecture which serves as a basis for the development of location-based MMMOGs. It is a sophisticated server backend structure to best serve the needs of location-based mobile games. Developers can use this architecture to build their individual applications on top of it.

We first present related work in Section II. In Section III we describe the general requirements for a MMMOG architecture and present our solution in detail. Section IV describes a game called Geobashing which was realized using our architecture. An evaluation of our work is presented in Section V.

II. State of the art

In the field of pervasive and mobile games, several projects have been proposed which partially include concepts similar to our Geobashing game. The properties of those games are useful to identify requirements for a general MMMOG architecture.

The Gopher game combines mobile gaming with user-generated content and additional Web 2.0 aspects [1]. Players can create tasks for other participants by placing virtual gophers somewhere in the world. Players nearby can pick up a gopher and try to fulfill the mission. A mission is composed of a textual description with optional additional media content. Once the player considers the mission complete, a jury (consisting of the other Gopher players) determines whether or not the goal was reached. The Gopher game uses the cell ID to determine the player's location.

An approach to migrate games to mobile platforms is a paper chase game developed by Boll et al [2]. This is a real-life implementation of a classical paper chase game where the player visits checkpoints and solve a riddle in order to receive the next waypoint location. This game uses GPS to locate the player. Riddles and additional gaming information are fetched from the game server.

Feeding Yoshi [3] tries to integrate playing a mobile game into the everyday life of the player. The goal is to find food for a creature named Yoshi. The localization in the game is performed by exploiting the WLAN infrastructures in cities where protected WLANs are Yoshis and open WLANs are used by the player to collect food. The WLAN access points are not used for communication with the game server. The player has to manually upload the game results.

GPS Mission [4] is already available on the market. Players create missions that other players must fulfill in order to earn virtual money and trophies. The GPS Mission client periodically sends the current user position to the game server and shows the surrounding map with mission waypoints and virtual items.

Tourality [5] is a GPS-based game where players can play alone or in teams. The goal is to reach certain waypoints as fast as possible. During a game the client application sends the current position to the game server and also displays the positions of the other participants.

In [6] a middleware for pervasive games is proposed. This system focusses on crossmedia games, i.e., classical computer games in a virtual world that are played with different client devices. In contrast, our architecture is designed for massive multiplayer games which are played in the real world, i.e., where the physical user position is included in the game state. Thus, it supports crossmedia

games but assumes that the client device has localization capabilities.

III. The Geobashing MMMOG Architecture for Location-Based Games

A reusable architecture for MMMOGs that can already be deployed to the market today must fulfill certain requirements.

Client-Server Architecture: We assume a *client-server* architecture since a peer-to-peer system is hard to realize on current mobile devices. Additionally, without a centralized infrastructure the global game state (e.g. highscores, player rankings) is hard to determine.

Scalability: In order to support a large number of users, the server infrastructure (game backend) must be scalable. This means that the game state can be shared by multiple application servers to reduce load and latency. Scaling a system always brings up the need for data synchronization to prevent inconsistencies.

Optimized for Positional Data: The architecture must be specialized for the processing of positional data. A specialized backend structure will greatly reduce latency. This affects the load balancing strategy as well as data synchronization. In addition, specialized data structures should be used at the application server.

Minimum Latency: To provide the maximum gaming experience to the user, the architecture must provide means to reduce latency.

Support for Offline Gaming: Games that do not require Internet connectivity during the whole game must be supported. For example game parts without interaction among users often do not need an active connection to the server. Since an open internet connection on a phone uses a lot of energy, the architecture must be able to deal with clients that are not constantly connected.

Use of Standard Technologies: To reduce development time for a mobile game and to facilitate cross-platform client development, it should be built upon proven standard technologies such as HTTP web servers.

Using the requirements stated above as a starting point we designed and implemented a reusable game architecture that we call the Geobashing MMMOG architecture.

A. The Geobashing MMMOG Architecture

The server infrastructure basically consists of a web service layer and an application server layer. The web service layer is only a thin entry point to the application server layer. The application servers perform all game-specific tasks. Figure 1 shows the Geobashing MMMOG architecture.

1) Web Service Layer: Since web services, or—more particularly—the web server they are running on, represent a major bottleneck in many web applications, their functionality is reduced to a minimum in our system. The main tasks of this component are user authorization and forwarding of requests to the application server. Due to this reduced functionality, a transparent load balancing of the web services can be realized easily, using any well–known load balancing mechanism (e.g. DNS-based load balancing). In order to support the application server load balancing described below, web services hold a mapping between users and responsible application server instances. This mapping is the only shared state that all web services have to access and is used to directly address the currently responsible application server for a user. If a mapping between a user and an application server cannot be found, the request is relayed to the root application server. To make this data available, a distributed hash table (DHT) is used.

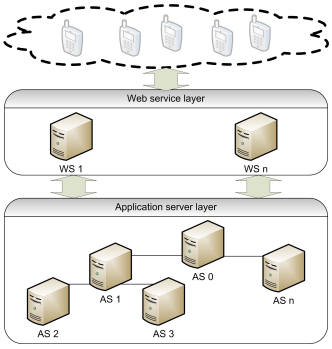

Figure 1. From a high level view the Geobashing architecture consists of a web service and an application server layer.

2) Application Server Layer: The application nodes implement the game logic and manage the game state. We assume the game state to be a set of static and dynamic game objects, which have a certain position on a map. Since heavy load on these nodes has to be expected, a load balancing technique must be implemented. In our approach the load on the application server instances is balanced based on the user's current position which is part of every user request. The server topology consists of multiple application nodes where every node is responsible for a certain geographical area (which we call the server's bounding box).

Since responsibility areas may be nested, we organize the application servers in a tree where the bounding boxes of all sub servers are contained in the root server's bounding box. To ensure a distinct mapping of user positions to servers, the bounding boxes of sub servers must not overlap. Every server knows its sub server but not its superordinate node. The advantage of this server infrastructure is that a server only maintains the state of game objects that are within its responsibility area and that are not contained in a sub server's bounding box. The server hierarchy provides the possibility to deploy a server infrastructure that addresses areas with a higher player density. If such an area is identified, the load can be split by adding more servers to the particular sub tree.

Like in every other application there are also requests that simply query data. In the context of mobile games these requests often enable the client application to provide parts of the game in offline mode (e.g. transfer challenges to the client). In the presented architecture, non-position-

based requests can be handled by any application node.

3) Game State: To support our architecture, we assume the game to have a lightweight game state that consists of only two object types: static and dynamic objects. Static objects are bound to a non-changing position on the map, whereas dynamic objects move on the map. Only static objects are stored in the persistent storage while dynamic objects are added on the fly.

4) Data Storage: The persistent storage is only accessed by the application nodes. The storage is used whenever static game objects need to be stored, modified or queried. Its main task is to provide the possibility to query static game objects based on a provided search area, e.g. whenever a new application server node is initialized.

5) Request Handling: Every application server processes a request in two phases (see Figure 2). In the *first phase* a sub server that is responsible for the request (i.e. the user position is contained in the sub server's bounding box) is searched. In *phase two* the request is either forwarded (if a suitable sub server was found; the sub server then performs the same query again) or handled by the current server. After handling the request, the response is returned through the server tree. This server selection process ensures that a request is always handled by the server with the smallest bounding box containing the user position. Since this lookup is an expensive operation, the response contains the ID of the server that actually handled the request. Thus, on the next request, the server can be queried directly.

The mapping between a user and the responsible application server is stored in the previously mentioned DHT in the web service layer. Since users move while being logged in, it is possible that the mapping between a user and a server becomes invalid. In this case the application server responds with a redirect message (see Figure 2), indicating that it is no longer responsible for the particular user. The calling web service then redirects the request to the root application node, a new responsible application node can be selected and the user-to-server mapping is updated.

A special type of request is the so called in-range request which queries game objects within a certain search area. In the application server infrastructure there is the possibility that this search area intersects with the responsibility areas of more than one server. In this case, the request is performed by the server that completely contains the search area. This server forwards the requests to all responsible sub servers and aggregates the data.

6) Latency Reduction: The presented load balancing mechanism is applied to distribute the load to multiple application servers within a single data center. By extending the web service layer, the same strategy can be used to achieve a distribution to different data centers. This addresses the need to reduce the latency on the client. In such a scenario the server infrastructure is present at multiple sites where every site has a different responsibility area. For this to work, the web service layer has to have knowledge of the global distribution and bounding

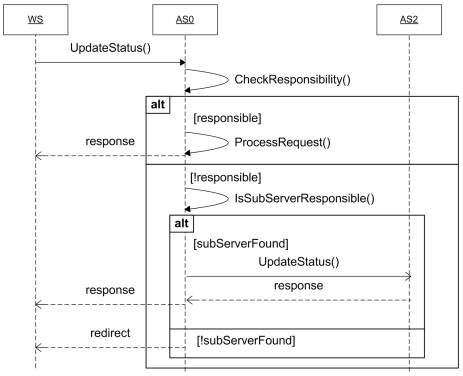

Figure 2. Selection of the responsible application server (simplified). WS is the calling web service.

boxes of the different sites. The lightweight game state supports such a multi-site infrastructure by partitioning the static objects according to their positions, as all objects are contained in a well-defined bounding box.

7) Fault Tolerance: The combination of the responsibility areas of servers, the possibility to redirect a request and the lightweight game state have an additional advantage. Due to this, a fault tolerant application server infrastructure can easily be realized. The system supports the transfer of the game state from one server to another without the need for a complex synchronization mechanism. If an application node fails, the web service layer will no longer be able to connect to this particular node. In this case the request will be forwarded to the root application node. If—at any point—a server cannot be reached, this particular server will be deleted from the parent's sub server list. Thus, the whole server sub tree will no longer be reachable. A user who was previously managed by a server in this particular sub tree will be assigned to a new server. Thus, all dynamic objects of the game state are migrated on the fly. The static objects can be easily added to the game state of the server now responsible by simply querying the data storage providing the bounding box of the failed sub tree.

8) Implementation Details: To support fast development of client applications, the implementation of the web service layer solely relies on Restful Web services (based on HTTP). The web service and the application server are implemented in .NET. For the communication between the web service and the application servers or between different application servers .NET Remoting is used.

To provide an easy way to exchange and extend the functionality of the application server, it is realized using the plug-in framework presented in [7], which is available as an open source project [8]. This enables game developers to use the Geobashing MMMOG architecture and build different game concepts on top of it. To support faster querying of position-related data at runtime, an R-tree [9] is used to store the game objects at the application servers. We use the R-tree implementation of Sharpmap [10]. For the DHT we use Memcached [11].

The default communication pattern in our architecture

is a client-side pull communication. Nevertheless, in some cases the server may want to explicitly notify the client of some event (to improve the game flow). For those cases we use a push communication channel based on SMS messages.

IV. The Geobashing game

As a use-case of our architecture we present the Geobashing game. Geobashing combines player interaction with mobility aspects, sports and role play elements. Players in this game can challenge other players with different tasks. By completing challenges, players earn experience points and virtual money. This part of Geobashing is called the *active part* (players have to actively choose to create or participate in a challenge).

In contrast to other games, Geobashing is meant to be played all day. Players can leave the application running in the background while carrying their phones. The application periodically sends the current GPS position to a server (this is called the *passive part*). The update frequency is controlled by the server which yields the possibility for server-controlled energy management based on e.g. player density. The server also evaluates if other players are nearby. Like in other role plays, a player can attack if nearby players are encountered.

A. Game World

The world of Geobashing is a virtual overlay to the real world. Every game element has a geographical position attached. A Geobashing player only needs to know the part of the virtual world that is immediately surrounding her current position.

B. Game Elements

A *character* is the virtual representation of the player and has the attributes *attack*, *defense*, *stealth*, *health*, a *level* and *experience points*, a purse for *money* and can carry *items*.

Items are static objects, i.e., they are bound to a certain position and do not move. Players can carry a certain number of items with them. They consume storage slots and can alter character attributes (e.g., the mighty sword of destruction, or the red baby-trike). They can be dropped or picked up by a player which enables *trading* between players.

Traps are similar to items, but additionally, they can be dropped and activated. If a player approaches a trap, a corresponding action is triggered (e.g., the player loses a specific amount of health points).

Money is needed for several tasks and is modeled as an item with a count.

Every player can define a certain position as *home base* for her character (e.g., her home). Within a fixed radius around the home base the player is not attackable. If a player is in this zone, she automatically restores health points. Additionally, she can exchange money and items with the home base storage or access the global Geobashing *item shop*. The home base can also be equipped with items which provide services to its owner or other players (e.g., a hospital: allows other players to restore health points for money).

C. Challenges

Players can participate in *Challenges*. The first time a player participates in a challenge a starting fee may be paid as a bet. A challenge consists of at least one position, has a goal and a reward. We have defined the following types of challenges so far.

A *Race Challenge* follows the ”be the fastest” principle. A track of waypoints must be passed and the player gets ranked according to her best time. On expiration, the sum of all starting fees is split among the best participants. Participants may receive medals, items and/or experience points as a reward. The creator may place a bet, which gets added to the rewards and she receives experience points for every participating player beating the creator’s reference time. If the participant does not pay the fee, she receives no monetary reward.

Another type is the *Paper Chase Challenge*. The idea is to give the character hints or let her solve riddles in order to get the next waypoint. Neither is the time relevant nor is a participation fee needed, but one can only participate once. A reward is optional. The creator is rewarded with experience points for every successful participant. A ranking is not needed for this type.

The *Shape Challenge* adopts the idea of GPS drawing[1]. A player who draws a given shape with the longest GPS track wins this challenge. Rewards and rankings are handled in the same way as in the race challenge.

For the most possible freedom players can create an *Open Challenge*. The creator only needs to declare a condition and a reward (e.g., ”Pick me up at noon, drive me to the next bar, I might have a cool item to give away.”).

The *Group Challenge* addresses groups of players. A group challenges is a composition of multiple single player challenges. Players form a group and find an approach to solve all challenges as fast as possible. All challenges must be successfully mastered by the group, whereas every player must participate in at least one challenge. The majority of the group must vote for the start of the group challenge (this starts the timer), it ends when all challenges have been attended at least once. The experience rewards are multiplied by a group factor and are distributed to the members, all other rewards are discarded. The best ranked players are awarded gold, silver and bronze *medals*.

D. Fight

Whenever two players are within a given range, they both are notified of the other (see screenshot in Figure 3). This range depends on the stealth attribute of each player character. The player can then decide whether or not to attack.

In a fight, the attacked player has the chance to escape. If, within a certain time frame, she can flee from the attacker (by overcomming a distance), the fight is aborted. Otherwise, if the attacker chases the attacked player and

[1] e.g. http://www.gpsdrawing.com

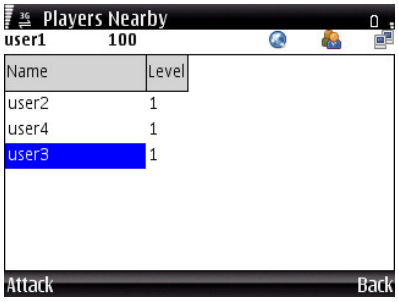

Figure 3. A screenshot of Geobashing showing the players in range. The three icons at the top right indicate (left to right) that GPS is online, players are in range and that the player is logged in.

the time frame expires, a fight is started. The fight is turn-based and the result is based on the individual attack and defense attributes of the characters. The fight result is calculated at the server without player interaction. However, the attacker can cancel the fight at any time. This also means that the attacked player may try to persuade the attacker to abort the fight.

E. Player Interaction

Geobashing itself defines no communication or interaction channels between players. This means that there is e.g. no messaging system. We assume that in a game, where players play in the real world, they also should interact like in the real world. To trade, players have to bargain for good prices for their items. To persuade an attacker to abort a fight, a player will have to be creative. We consider this a very essential part of the Geobashing game.

F. Interfaces

Geobashing provides a *mobile client* and a *web application*. The mobile client enables the player to participate in challenges. It also periodically reports the current position to the server and notifies the player of items or other players in range. Using the web application, a player can manage her home base, download challenges to the mobile client, view her statistics and challenge details.

V. Evaluation and Results

To evaluate our architecture, we performed several tests with the Geobashing game. The Geobashing game is not very sensitive to high latencies, but to preserve a good user experience, latencies below five seconds have proven to be acceptable. For games that are more sensitive to latency it is recommended to either load balance the application servers (AS) or to bring web services closer to the player.

In a first experiment we tested how latencies change if a greater load is posed on the server. To achieve this we simulated varying numbers of players that continuously perform the status update request. In this request the client application submits the current location, the server searches for surrounding players and game objects and returns this list to the client. In this test we used an Intel Core i7-920 based server with 8 GB of main memory. The web service (running on Mono XSP2) as well as the single AS were running on the same machine. The server was running Debian with Mono 2.4.4. We implemented a Geobashing client prototype using the .NET Compact Framework.

#Simulated Users	*Avg. client latency[ms]*	*Avg. Processing Time[ms]*
0	653	15.8
1,500	704	21
7,500	570	37

Table I
THE LATENCY AT THE CLIENT AND THE PROCESSING TIME PER REQUEST WRT. VARIOUS USER NUMBERS.

We measured the time required to execute the request on cell phone clients (Nokia E71 running Redfivelabs Net60[2] runtime for .NET Compact Framework in the cellular network of A1 Austria). Additionally, we measured the request duration at the server to be able to assess the latency of the mobile network. The results of this experiment are summarized in Table I.

The difference between the time measured at the clients and the actual processing time at the server shows that the latency at the client is mostly caused by the mobile network. In this experiment our server was located in Germany, the clients were running in Klagenfurt, Austria. Thus, by bringing the web service closer to the client, a reduction of latency can be achieved. This experiment also showed that the processing time per request at the server increased with the number of users. Since we did not simulate the full range of requests a typical user would perform, we expect a higher load with the same number of real clients. This in return, indicates the possible need for load balancing the AS at a number of concurrent users below 10,000.

In a second experiment we evaluated the maximum number of users that a single AS can handle. We used a lab environment and split application server and web service to separate machines. The AS was an Intel Core2Duo 1.86 GHz with 2 GB main memory. All servers were connected using 100MBit/s Ethernet and ran the Mono 2.6.1 LiveCD (based on OpenSuse) with Mono XSP2 as web server. Again, clients were simulated to generate the necessary load.

The experiments showed that the AS can handle more than 500 requests per second without significant increase of latency. This roughly corresponds to 5,000 to 8,000 (simulated) users. Additionally, this experiment showed that the web service layer has a higher risk of being a bottleneck.

Finally, we evaluated the presented load balancing mechanism. We focused on the costs of redirects and forwards. In both cases the request cannot be posed directly to the AS but—starting from the root AS—the responsible server must be found in the application server tree. In case of a forward, the responsible server could not be found in the distributed hash table (DHT). In case of a redirect, the server was found but it denied service because the user had left the server's responsibility area. Thus, a

[2]Unfortunately, the company recently went out of business.

redirect is preceded by one unsuccessful request. If this cost is very high, players that migrate from one AS to a neighboring would generate a high overhead. Additionally, we tested the performance of in-range requests that cannot be handled by a single server (as described in Section III-A5).

Using the same lab setup as in our second experiment, we defined the following setup of application servers: AS1 is the root AS responsible for the Austria region. AS2 is responsible for the players in Vienna. AS3 covers southern Austria (Styria and Carinthia) and AS4 covers Klagenfurt and the Woerthersee region. Thus, AS2 and AS3 are contained in AS1 and AS4 is contained in AS3.

Table II shows the time necessary to find the responsible AS in the application server tree in case the player was not present in the DHT. The maximum number of redirects within the application server tree is two in our setup (for AS4; redirect from AS1 to AS3 to AS4). Zero requests means that the root application server (AS1) was responsible for the player. Table III shows the search time in the application server tree in case of a redirect. Zero requests means that after the initially failed request to a sub server (AS2 or AS3), the application server was responsible for the player.

# Requests	*Avg[ms]*	*Max[ms]*	*Min[ms]*
0	3.05	2189	1.3
1	6.13	4124.7	2.7
2	7.14	2351.2	4.1

Table II
THE DURATION TO FIND THE RESPONSIBLE AS IN CASE OF FORWARDING.

# Requests	*Avg[ms]*	*Max[ms]*	*Min[ms]*
0	5.90	1851.5	3.55
1	8.12	1015.25	4.98
2	10.92	509.45	6.31

Table III
THE DURATION TO FIND THE RESPONSIBLE AS IN CASE OF REDIRECTING.

The results show that forward and redirect differ mainly in the time for the single initial request of the redirect. A direct query to an AS takes about two to three milliseconds in this setup and the search for a suitable server takes linear time. By keeping the Player-to-AS mapping in the DHT, in most cases the responsible AS will be queried directly.

To test the performance of the multi-server in-range requests we simulated users in the border region of AS4. To find all neighboring players, AS3 had to merge its own results with the results of AS4.

The average single-server request at AS3 took 0.26ms while the average time to perform the multi-server request was 3.5ms. The additional request to AS4 again took approximately 3ms. To save time, a multi-server request with a higher number of servers involved is performed in parallel. In the worst case, four servers must be queried to perform this request. This shows that the relative cost of a multi-server request is high but judging from the absolute value, this will not influence the game flow. Furthermore, the radius in which Geobashing searches for neighboring players is only 200m, i.e. this case is unlikely to occur often in the Geobashing game.

VI. CONCLUSION

In this paper we presented an architecture for MMOGs. This architecture can be used as basis to build such games. The architecture relies on proven technologies like HTTP and .NET Remoting. The web service and application server layers have been decoupled and can be load balanced as needed. The game-relevant parts can easily be exchanged to build new games on top of the architecture.

We also presented the Geobashing game which combines sports with social interaction and contains role play elements.

We evaluated our architecture with the use-case of the Geobashing game and showed that the architecture is suitable for a massive amount of players.

In the current version, the application server tree structure is preassigned manually. In future work, we will include mechanisms to dynamically change the application server tree for dynamic load balancing and latency reduction.

REFERENCES

[1] S. Casey, B. Kirman, and D. Rowland, "The gopher game: a social, mobile, locative game with user generated content and peer review," in *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2007, pp. 9–16.

[2] S. Boll, J. Krösche, and C. Wegener, "Paper chase revisited: a real world game meets hypermedia," in *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*. New York, NY, USA: ACM, 2003, pp. 126–127.

[3] M. Bell, M. Chalmers, L. Barkhuus, M. Hall, S. Sherwood, P. Tennent, B. Brown, D. Rowland, S. Benford, M. Capra, and A. Hampshire, "Interweaving mobile games with everyday life," in *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. New York, NY, USA: ACM, 2006, pp. 417–426.

[4] "GPS Mission," http://gpsmission.com.

[5] "Tourality," http://www.tourality.com.

[6] F. Trinta, D. Pedrosa, C. Ferraz, and G. Ramalho, "Evaluating a middleware for crossmedia games," *Comput. Entertain.*, vol. 6, no. 3, pp. 1–19, 2008.

[7] B. Dieber, B. Rinner, and N. Viertl, "Flexible Clustering in Networks of Smart Cameras," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, October 2009, pp. 834–839.

[8] "NIPO Framework," http://nipo.codeplex.com.

[9] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1984, pp. 47–57.

[10] "SharpMap," http://sharpmap.codeplex.com/.

[11] "Memcached," http://www.memcached.org.