An Optimized Location-based Mobile Restaurant Recommend and Navigation System

ZHI-MEI WANG
Department of Computer
Wenzhou Vocational Technical College
Wenzhou, 325035, Zhejiang
CHINA
wzmfst@126.com

FAN YANG
Continuing Education College
Shanghai JiaoTong University
No. 1954, HuaShan Road, 20030, Shanghai
China
fyang@sjtu.edu.cn

Abstract: -With the widely used of the intelligent mobile phones with the GPS, the location-based services has become the a hot issue of mobile communications research. This paper implements a Mobile Location-based Restaurant Navigation and Recommend System. In order to improve server-side response speed for real-time query, we propose a memory pool model, the expansion Accept command, no-data client polling and interrupt mechanism, which aims to greatly optimize the server-side control procedures. On the client side, we combine the latest Web2.0 application data with the location-based data, and propose a collaborative assessment and recommend mechanisms, which can provide users with real-time location-based restaurant and recommend personalized navigation.

Key-Words: Mobile Information Share, GPS, Web2.0, Location Based Service (LBS), Tagging, Collaborative Filtering, Personalized Recommendation

1 Introduction

Nowadays, the intelligent mobile phones with the GPS functional component become very popular and widely used. How to provide timely and personalized information and sharing services based on the user's location information? This problem is gradually contracting wide range of concerns of different areas of the researchers, content providers and network operators. And it forms a known and independent research area named as Location Based Services (LBS) [1-2].

Location-based services (LBS), is the use of certain technical approaches through the mobile network to access the end-user’s location information (latitude and longitude coordinates), and provides users with a corresponding value-added Services through the electronic map platform [1-2].

The new generation of multimedia mobile phone, like iPhone, has begun to integrate online LBS services as Google maps to help users access to their destinations with traffic information and road conditions.

LBS is the integrated business of mobile network and location-based services, which aims at providing location and personalized information services to frequently location changing mobile users.

Location and context are the core of LBS. Thus LBS is also known as Location-Aware Computing, or Context-Aware Services.

Compared with traditional Geographical Information System (GIS) [3], from the hardware and software perspective, LBS is

** This paper is partly supported by the project founded by WenZhou Science & Technology Bureau of China [Project No. H20080008], and the projec“Research and Implementation on Web2.0 Information Sharing Platform based on GPS Services”founded by Science and Technology Department of Zhejiang Province in 2009.

involved in more platforms and components, including the Internet, GIS, positioning equipment and telecommunications technology and so on.

From the data perspective, LBS needs to obtain data from different sources, such as remote sensors, positioning systems, electronic maps, traffic and transportation databases and so on.

Therefore, from the system architecture perspective, LBS has a strong heterogeneity. At the same time, the user's location is constantly changing. Thus, the data-processing capability in the server side LBS services on the system server-side has brought new challenges [4-6].

For this new type of location-based information retrieval approach, users want to be able to obtain more real-time and targeted content services, not just the indexed information based simply on a static database[7-8]. Recently, the rise of a large number of Web2.0 applications (blog, community forums, Web Albums, Blog and Taggings, etc.) indicates that users have the very pressing requirements of direct, rapid, useful and personalized information recommendation and sharing services [9-13].

If the information can be user-friendly visualized in the client mobile terminals, It should doubtless be a very important research topic, and will have a very wide market prospect.

This paper designs and realizes a location-based mobile restaurant recommendation and navigation system. In order to improve server-side response speed for real-time query, we propose a memory pool model, the expansion Accept command, no-data client polling and interrupt mechanism, which aims to greatly optimize the server-side control procedures. On the client side, we combine the latest Web2.0 application data with the location-based data, and propose a collaborative assessment and recommend mechanisms, which can provide users with real-time location-based restaurant and recommend personalized navigation.

Users can also manually provide personalized tagging and recommendation to build their own social networks, which can help them to consider other similar community users'collaborative comemnts and obtain more presice content pushing service.

Section 2 presents a simple description of the system's overall architecture and component. The server-side operating mechanism, working threads, listening thread mechanism and optimize the statement is discussed in Section 3. And in section4 you can find the introduction of the functional in the client side considering the users commend and recommend mechanisms . A case study is carried out in Section 5. Finaly, the conclusion of this paper and future work overview are discussed in Section 6.

2 System Workflow and Architecture

Figure 1 gives the workflow of our system. Users can send their inquiries demand by operating in the mobile phone. And the client will get the current location information and sent it together with users' inqueries demand to the server. Server-side application will analyze the relevant data and provide matched restaurant recommendation and navigation.

Application data information of our system can be divided into two parts: the location-based data (such as traffic and road condition data, GPS map, and entity information, etc.) and the value-added data provided by users (such as Ratings, Comments, Blog and Tags, etc.).

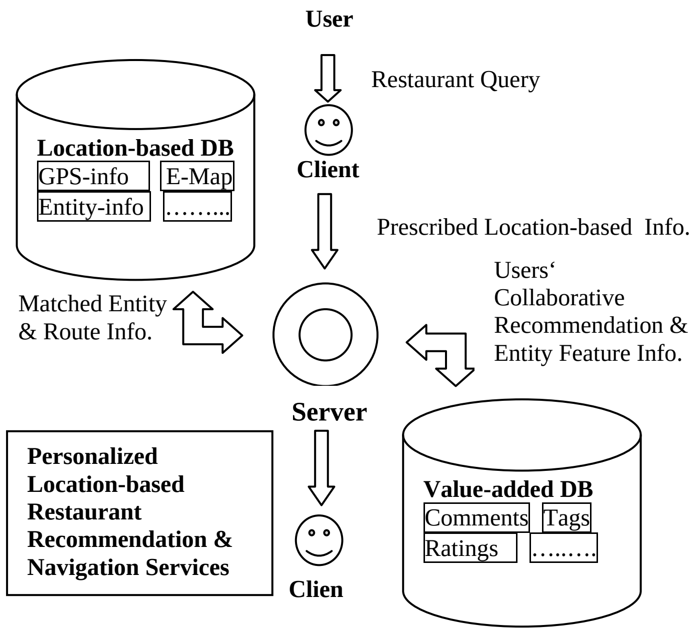

Fig.1. System Workflow

The system will obtain tbe initial restaurant candidates through the matching in location-based database (such as distance from the current location of 500 meters radius) restaurants navigation information. Furthermore, the system will be coordinated to analyze the user's comments information and refilter the initial candidates, thus return the restaurants more fitting users' requirements.

In order to know users' acceptance of our recommendations, we propose the 'mobile discount coupons' which can be directly used when the users show it to the restuarant. Through the usage of the 'mobile counpons', we can analyze the users' interest and characters, which can help us to effectively improve the accuracy of recommendation. The application data shows our system can enhance the acceptance and usage of mobile coupons which do benefit the users and companies simultaneously.

3 Server-side Implementation and Optimization

Our Server operating systems is based on Windows server 2003. The reason to choose is because its completion port (IOCP) technology is basically considered in the windows operating system as most sophisticated and efficient methods of IO. The overlap I/O technology by using the IOCP provide a real scalability for windowsNT and windows2000. Combining with the Windows Socket 2.0, it can develop the network services which can support a wide range of connect procedures.

3.1 Working Thread

Working thread is the most central part of the server and it is closely related to the server efficiency, stability, etc. Figure 2 gives the workfolw of the working thread. This thread primarily obtains the status of the client socket through the *GetQueuedCompletionStatus* function, which can be called as (OPERATION_ACCEPT, OPERATION_RECV, OPERATION_SENT).

Here OPERATION_ACCEPT Indicated that a new client connection requests to come in. Then the new socket will be bound to IOCP and make a request to receive data (PostRecv).

OPERATION_RECV means that data has come in, and access to the client operation orders according to custom head protocol information. Then it will make the corresponding treatment, and return the results back to the client.

OPERATION_SENT shows the completation of data sending from server-side. It can continue to further send operations.

When there is an error occurs, it will turn off the corresponding client socket, recover memory, and take the task from the queue and put into idle queue.

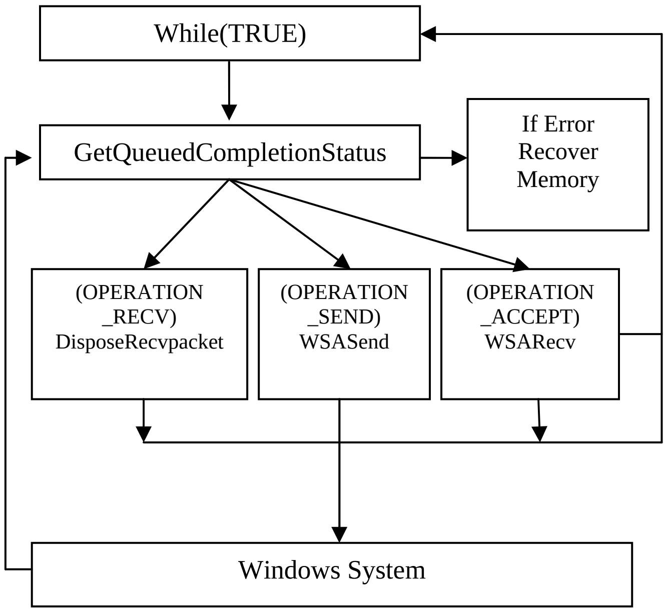

Fig. 2. Working Thread Workfolw

3.2 Listening Thread

The Listening Thread will create a new event, linked it with FD_ACCEPT through WSAEventSelect then wait for this event using WaitForSingleObject function. When the number of AccpetEx calls have been depleted and there are new client which require to connect, the FD_ACCEPTEvents will be triggere. If the status of the EVENT has been turned into be-sended, and return the WaitForSingleObject, then we will resend enough AcceptEx calls.

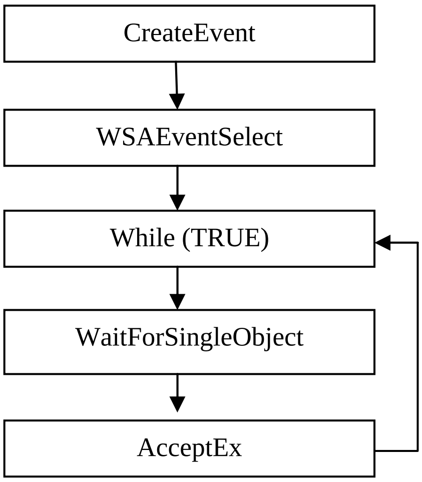

Fig. 3. Listening Workflow

3.3 Optimization Mechanism

In order to improve real-time query response in Server-side, we made a series of optimization mechanisms:

1) Memory pool model

First of all, we take use of the memory pool model and set up to four queues including: m_lpBusyPerHandleData(the in-use single-handle data), m_lpIdlePerHandleData(the free single-handle data), m_lpBusyPerIoData(the in-use single-IO data)□m_lpIdlePerIoData(the free single-IO data).

When to apply for a new data, the system will first check whether there is any queue which has available space. If so, it will take out a procedure for use, and put it into the in-user queue, Otherwise, it will ask for a space and add it to the in-use queue.

When a client leaves or an error occurs, it will recover the memory immediately and put the procedure into the free queue for the next use. This technology can effectively improve the memory utilization, reduce memory fragmentation and accept more client connections, thereby increase the server capacity and processing speed.

2) AcceptEx procedure

We propose a extend procedue AcceptEx insteadof the traditional Accept procedure. In thi procedure, it must bind the listening socket m_hListenIt with FD_ACCEPT. During the procedure, we should call the Socket or WSASocket function and create a new socket.Then this new socket can be passed to the AcceptEx function through the parameter *sAcceptSocket*, which can finally accelerate the speed of accepting the client.

3) No-Data client polling and interrupt mechanism

Every 3 seconds, the server will poll all client sockets. If there is no data transmission in more than three seconds, the socket will be considered overtime, and the server will disconnect it. This mechanism can save the server resources to the greatest possibility and provide services for more clients.

4 Client-side Implementation

Client users can simply enter search keywords and fuzzy constraints(such as the surrounding distance, food tastes, grade, etc.). And the server will feed back the matched restaurant information as follows

1) Basic information: including name, telephone, address, recommend dishes, brief introduction, the per capita consumption as well as the classification.

2) Collaborative recomemndation information: recommendation based on the collaborative filtering of other users' tagging, rating, commends data.

3) E-map and navigation: restaurant with a balloon-shaped signs displayed on the vector map, and can real-time navigation.

4) Restaurant coupons: name, preferential margins, maturity dates and coupon bar code.

Client implementation major includes the design and de development of Functional Class and View Class described as follows:

4.1 Functional Class in client-side

The client-side includes six Functional Classes: *CstaticImageDecoder, CtransEngine, CClientEngine, CMyListBox, CmyPicture, and CsocketsEngine.*

4.1.1 CstaticImageDecoder Class

Inherited from the Class *CActive*, this class realizes the asynchronous decoding of compressed image. It uses *CBufferedImageDecoder* to decode the gif image and feedback the decoded bmp to the caller as shown in Figure 4.

When there is no decoding error, it will call the *DecodeComplete* function of *MdecoderNotifier* to notify the caller that the image decoding process is finished.

FoodSearch::**CStaticImageDecoder**
-iDecoder : CBufferedImageDecoder * -iNotify : MDecoderNotifier &
<u>+NewL(inout aNotify : MDecoderNotifier) : CStaticImageDecoder *</u> +CStaticImageDecoder(inout aNotify : MDecoderNotifier) +~CStaticImageDecoder() +StartDecode(inout aData : TDesC8) -ConstructL() -RunL() -DoCancel()

Fig. 4. *CstaticImageDecoder* Class

4.1.2 CtransEngine Class

CtransEngine Class is mainly in charge of the communication between the client-side and server-side. Since the system needs to maintain only one connection, this class takes use of the Singleton design mode.

According to different callers' Notify types, the *CtransEngine* Class has two callers as EshopList and EshopDetail shown as follows:

```
int
CTransEngine::ConvertFromGB2312ToUnicode(TDes16& aUnicode, const TDesC8& aGb)
{
TIntstate=CCnvCharacterSetConverter::KStateDefault;
TInt ctu = iConverter->ConvertToUnicode (aUnicode, aGb, state);
if ( ctu ==
CCnvCharacterSetConverter::EErrorIllFormedInput)
User::Leave (KErrCorrupt);
return 0;
}
```

4.1.3 CclientEngine Class

CclientEngine class is also related to network communication. However it deals with the Http request different the *CtransEngine* class.

4.1.4 CmyListBox Class

CmyListBox class inherites from the *CeikTextListBox* class and realize the user-define list as shown in Figure 5. The functions of the user-defined list includes: add or delete the list item, change the list item's height, change the overall size of the components, change the background and highlight background, characters and icons display forma of a single list. Similar with any other Symbian components, the list takes use of the MVC (Model - View - Controller) model.

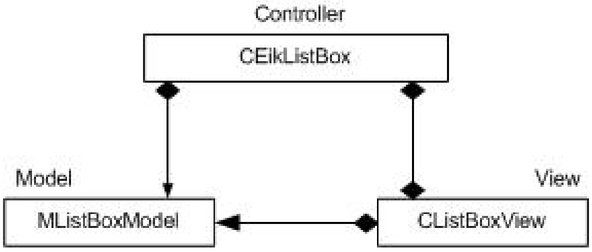

Fig.5. MVC model of Symbian List

4.1.5CmyPicture Class

CmyPicture inheriates from Cpicture class which is mainly used to display imags in RichTextEditor.

4.1.6 CsocketsEngine Class

CsocketsEngine class is responsible for setting up connections between local socket and remote socket, amd implement the DNS search (if necessary). It will also create an instance of *CsocketsReader* and *CSocketSWrite* active object, which can control the receiving and sending process of asynchronous data.

The initial state of the CSocketsEngine class is set as EnotConnected. It will build a counter to ensure the failure of asynchronous request which did not finish within the limited time duration shown as follows.

```
void CSocketsEngine::ConstructL()
{
ChangeStatus(ENotConnected);
iTimer = CTimeOutTimer::NewL(EPriorityHigh, *this);
CActiveScheduler::Add(this);
User::LeaveIfError(iSocketServ.Connect());
iSocketsReader = CSocketsReader::NewL(*this, iSocket);
iSocketsWriter = CSocketsWriter::NewL(*this, iSocket);
}
```

The communication process between the client and remote server is asynchronous. We specified the remote server IP address and port number, and connect them.

```
iSocketsEngine->SetServerName(serverName);
iSocketsEngine->SetPort(port);
iSocketsEngine->ConnectL(); // Initiate connection
```

Client requests a way to achieve asynchronous write operation, and use the CSocketWriter class to dispatch these requests.

CsocketWriter Class uses a Buffer (iTransferBuffer)to accept the buffer from UI (iWriteBuffer). And we use the CSocketsEngine::WriteL function to send the characters to the engine shown as follows.

```
void CSocketsEngine::WriteL(const TDesC8& aData)
{
        // Write data to socket
        if (iEngineStatus == EConnected)
        {
            iSocketsWriter->IssueWriteL(aData);
        }
}
```

The request can be sent to CsocketWriter through the CSocketWriter::IssueWriteL Class shown as follows.

```
    void CSocketsWriter::IssueWriteL(const TDesC8&
aData)
    {
         if ((iWriteStatus != EWaiting) &&
(iWriteStatus != ESending)){
                User::Leave(KErrNotReady);
         }
         if ((aData.Length() +
iTransferBuffer.Length()) >iTransferBuffer.MaxLength())
         {
         User::Leave(KErrOverflow);
         }
         iTransferBuffer.Append(aData);
         if (!IsActive()){
             SendNextPacket();
         }
}
```

When the data is copied into the transfer buffer, the system will call the CSocketWriter::SendNextPacket function shown as follows.

```
void CSocketsWriter::SendNextPacket()
{
        if (iTransferBuffer.Length() > 0)
        {
        iWriteBuffer = iTransferBuffer;
        iTransferBuffer.Zero();
        iSocket.Write(iWriteBuffer, iStatus); // Initiate
actual write
        iTimer->After(iTimeOut);
        SetActive();
        iWriteStatus = ESending;
        }
        else
        {
        iWriteStatus = EWaiting;
        }
}
```

CSocketsWriter::SendNextPacket function will remove the data into the write buffer, clear the transfer buffer and call the RSocket::Write function to send the data request. The status of the CSocketsWriter object will be changed into Esending, and start the counter to control the status of the RSocket::Write. When the request of RSocket::Write is finished, the system will recall the CSocketsWriter::RunL function shown as follows.

```
void CSocketsWriter::RunL()
{
        iTimer->Cancel();
        if (iStatus == KErrNone);
        {
            switch(iWriteStatus)
            {
                case ESending:
                SendNextPacket();
                break;
                default:
                User::Panic(KPanicSocketsEngineWr
ite, ESocketsBadStatus);
                break;
            };
        }
        else
        {
            iEngineNotifier.ReportError(MEngineNoti
fier::EGeneralWriteError;
            iStatus.Int());
            iWriteStatus = EWaiting;
        }
}
```

The Definition of this class can be shown in Figure 6.

Fig. 6. CSocketsEngine Class

4.2 View Class in Client-side

The View classed in Client-side is composed of five classses as : *CfoodSearchContainer,* *CshopListContainer*, *CshopDetaiContainer*, *CfoodSearchMapViewContainer*, and *CcouponContainer*.

```
iSocketsEngine->SetServerName(serverName);
iSocketsEngine->SetPort(port);
iSocketsEngine->ConnectL(); // Initiate connection
```

4.2.1 CfoodSearchContainer Class

This class includes a CeikEdwin and a button. CeikEdwin is used to accept user's input and converts the encoded input as a parameter passed to the category *CshopListView* class.

This Container includes a CeikEdwin and a button. Here, CeikEdwin is used to accept user's input and recode the input content as a parameter sent to the CshopListView Class. Partial recall and transfer codes can be shown as follows:

```
TBuf8<100> temp8;
	CCnvCharacterSetConverter* iUnicode2GbConverter = CCnvCharacterSetConverter::NewL ();
	CleanupStack::PushL(iUnicode2GbConverter);
	CCnvCharacterSetConverter::TAvailability
	ta = iUnicode2GbConverter->PrepareToConvertToOrFromL (
	KCharacterSetIdentifierGbk, CEikonEnv::Static()->FsSession ());
	if ( ta != CCnvCharacterSetConverter::EAvailable)
		User::Leave (KErrNotSupported);
	CleanupStack::Pop(iUnicode2GbConverter);


	TInt state=CCnvCharacterSetConverter::KStateDefault;
	TInt ctu = iUnicode2GbConverter->ConvertFromUnicode (temp8, buf, state);
	if ( ctu == CCnvCharacterSetConverter::EErrorIllFormedInput)
		User::Leave (KErrCorrupt);
delete iUnicode2GbConverter;
	iUnicode2GbConverter = NULL;
	STATIC_CAST(CFoodSearchAppUi*,CCoeEnv::Static()->AppUi())->ActivateLocalViewL(TUid::Uid(EShopListContainerViewId),TUid::Null(),temp8);
```

4.2.2 CshopListContainer Class

CshopListContainer class is incharge of display the restaurant list. In this class, it obtain the restaurant information from the remote server and display the results as user-defined list. Figure 6 shows the sequence of restaurant list acceptance.

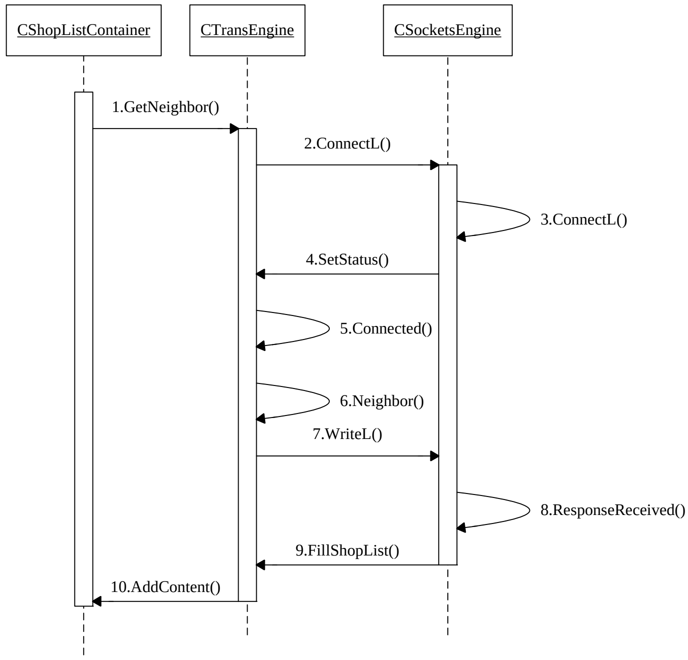

Fig. 7. The sequence of restaurant list acceptance

4.2.3 CshopDetaiContainer Class

CshopDetaiContainer class can display the restaurant's information. It obtains the information from the remote server through the CtransEngine, and displays the results throug RichTextEditor. Furthermore, through the creation of CmyPicture class, we also realize to display some images in the RichTextEditor in order to make the restaurant introduction more lively. Restaurant information also includes the invisible GPS information and Restaurant ID. These two parameters are separately transmitted as the parameters to the *CCouponView CFoodSearchMapView* classes in charge of map visualization. Figure 7 gives the structure of *CshopDetaiContainer* class.

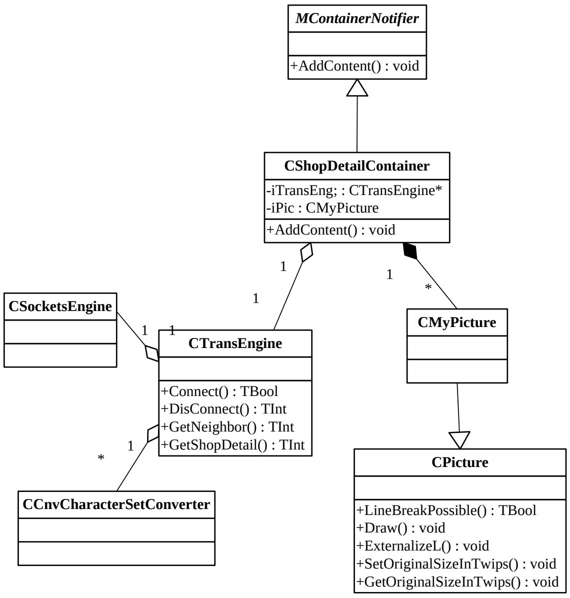

Fig. 8. Structure of CshopDetaiContainer Class

4.2.4 CfoodSearchMapViewContainer Class

CfoodSearchMapViewContainer class is used to displaythe map information of restaurant. It will calculate the latitude and longitude information and submit the result to the remote server.

4.2.5 CcouponContainer Class

CcouponContainer class can display the restaurant counpons, which including the restaurant name, coupon expiration time, the preferential margin, coupon ID, as well as bar code.

5 Case Study – A Dynamic Mobile Location-based Restaurant Navigation and Recommend System

Based on our platform, we cooperate with some restuarants to develop develop a dynamic restaurant mobile location-based recommendation and discount counpons pushing system. Based on our dynamic location-based resturant recommendation and navigation services, the user can easily find the resturant in a certain range of current location as shown in Figure 8.

Fig. 9. Restaurant Location-based Information

Especially, through this application platform, users can not only receive the static description of the restaurants which are suitable for their own tastes (such as size, styles, features, environment, etc.), but also can see the dynamic synergy of the community users tag information (such as ratings, comments, recommend dishes, etc.) as shown in Figure 9.

Fig. 10. Restaurant Information

Furthermore, we also provide a "mobile discount coupons" which can be directly used when the users show it to the restuarant as showin in Figure 10.

On the one hand, the use of mobile coupons can help us to know users' acceptance of our recommendations. On the other hand, through our collaborative filtering and personalized recommendation algorithms, our system can effectively improve the accuracy of recommendation which may satisfy the users and then effectively improve the acceptance of mobile coupons. The application data shows our system can enhance the acceptance and usage of mobile coupons which do benefit the users and companies simultaneously.

Fig. 11. Restaurant Counpons

6 Conclusion

This paper implements a Mobile Location-based Restaurant Navigation and Recommend System. In the server-side, we propose a series of opertimazation mechanism as memory pool model, the expansion Accept command, no-data client polling and interrupt mechanism, which aims to enable the server to have greatly capacity and response speed for real-time query. In the client-side, we combine the latest Web2.0 application data with the location-based data, and propose a collaborative assessment and recommend mechanisms, which can provide users with real-time location-based restaurant and recommend personalized navigation. We also give the detailed description of the funtion classes and view classed in client-side. Finally we propose the case study of our mobile location-based restaurant navigation and recommend system, which already has successful business application in China.

References:

[1] Koeppel, I. What are location services? From a GIS Perspective, ESRI white paper.2000.

[2] Shiode, N., Li, C., Batty, M., Longley, P., & Maguire, D. The impact and penetration of location-based services. In H. A. Karimi & A. Hammad (Eds.), Telegeoinformatics: location-based computing and services, 2004, pp. 349–366, CRC Press.

[3] Jiang, B., Yao, X. B. Location-based services and GIS in perspective. Computers, Environment and Urban Systems,Vol.30, No.6, 2006, pp. 712-725.

[4] Kaasinen, E. User needs for location-aware mobile services. Personal and Ubiquitous Computing, Vol. 7, 2006, pp.70–79.

[5] Csinger, A. Users models for intent-based authoring. Dissertation, The University of British Columbia, 1995.

[6] Karimi, H. A. Telegeoinformatics: Current trends and future direction. In H. A. Karimi & A. Hammad (Eds.), Telegeoinformatics: location-based computing and services, 2004, CRC Press.

[7] Li, C. User preferences, information transactions and location-based services: A study of urban pedestrian way finding. Computers, Environment and Urban Systems, Vol.30, No. 6, 2004, pp.726–740.

[8] Ashbrook, D., & Starner, T. Using GPS to learn significant locations and predict movement across multiple users. Personal and Ubiquitous Computing, Vol.7, 2003, pp. 275–286.

[9] Fan Yang, Bernd Kraemer, Zhimei Wang, An Novel E-Learner Community Construction Algorithm in Point View of Learning Interests, WSEAS Transactions on Computers, Vol.5, No. 12, 2006, pp.3097-3103.

[10] Zhi-mei Wang, Ling-Ning Li, Enable collaborative learning: an improved e-learning social network exploiting approach, Proceedings of the 6th Conference on WSEAS International Conference on Applied Computer Science, Vol.6, Hangzhou, China, 2007, pp.311-314.

[11] Zhimei Wang, Peng Han, Fan Yang, E-Learner Community Exploiting Based on Collaboration Filtering, Proceedings of the 6th WSEAS International Conference on Applied

Informatics and Communications, Elounda, Greece, 2006, pp.250-253.

[12] Fan Yang, Bernd Kraemer, Zhimei Wang, Peng Han, An Improved E-learner Community Construction Algorithm Based on Learning Interest Feature Vectors, 6th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC'06), Greece, 2006, pp. 254-259.

[13] Zhimei Wang, Peng Han, Fan Yang, Distributed E-Learner Community Discovery Based on Collaborative Filtering, WSEAS Transactions on Information Science and Applications, Vol.3, No.11, November 2006, pp.2239-2244.