International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:11, 2009

Free-Form Query for Cell Phones

R. Ahmad, and S. Abdul-Kareem

***Abstract*—**It is a challenge to provide a wide range of queries to database query systems for small mobile devices, such as the PDAs and cell phones. Currently, due to the physical and resource limitations of these devices, most reported database querying systems developed for them are only offering a small set of pre-determined queries for users to possibly pose. The above can be resolved by allowing free-form queries to be entered on the devices. Hence, a query language that does not restrict the combination of query terms entered by users is proposed. This paper presents the free-form query language and the method used in translating free-form queries to their equivalent SQL statements.

***Keywords*—**Cell phone, database query language, free-form queries, unplanned queries.

I. Introduction

ENABLING querying of information from remote databases anytime anywhere has become increasingly important for today's highly mobile society. However, for developers to build a database querying system for small mobile devices that can support a large range of possible queries is rather a challenge. Since these devices such as the Personal Digital Assistants (PDAs), palmtops and cell phones are known to be limited in terms of their physical, resource and networking capabilities [1], majority of the database querying systems developed for them are solely dedicated to their intended applications. Hence, possible queries that can be formulated on these systems are mostly being pre-determined by the application developers as sets of options provided on a menu such as the systems presented in [2],[3] and [4]. However, such a measure tends to limit the usage of these systems. It leaves no room for users to issue other queries than those given. Furthermore, this approach of accepting only precise queries also hinders such systems from being easily adopted for other databases or applications.

Therefore, a database querying system for mobile devices should be able to allow users to pose any query that they want. And, this has to be done using as minimal resources as possible. Thus, in this paper, a query language which is free-formed will be introduced and explained. Using the language, users can combine any database schema term, i.e., relation name and/or attribute name, of their choice in any particular order to form queries. Hence, no pre-determined queries will need to be provided by the developers.

The remainder of this paper is organized as follows. Section 2 highlights some related works, Section 3 introduces the free-form query language, Section 4 presents the translation process of converting free-form queries to relevant SQL statements, and Section 5 provides conclusions.

R. Ahmad is with the department of Computer and Information Sciences, Universiti Teknologi PETRONAS (UTP), Malaysia (phone: 605-3687477; fax: 605-3656180; e-mail: rohiza_ahmad@ petronas.com.my).

S. Abdul-Kareem is attached to the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia (e-mail: sameem@um.edu.my).

II. Related Work

Database querying has been the focus of many database researchers for a long time. However, the capability of transacting queries while on the move using small mobile devices has only recently gained interest from the database community. Currently, the above interest is mainly targeted to mobile devices of considerable resources such as the PDAs and the palmtops. Even for these devices, the works reported are mostly application specific. For example, Hung and Zhang [2] presented a telemedicine system which can be used to access patient general information and medical conditions such as blood pressure (BP) reading and ECG diagramming on PDAs. Meanwhile, Koyama et al. [3] developed a system for education application. Their system can be accessed on PDAs by students who want to perform lesson's unit test. Boonsrimuang, Kobayashi and Paungma [4], on the other hand, presented a system for transportation application, also on PDAs. Even though the above applications and others are undeniably important, they are very limited in terms of their functionalities. In other words, their usefulness is confined to a single domain of application, and even within that particular domain, they are restricted to the functions (queries) that have been pre-defined by their developers. Thus, there is no possibility for unplanned queries to be formulated on such systems.

By using precise input query method as the above, unplanned queries are rather hard to implement since they require almost all combinations of possible query terms to be thought of beforehand by the developers. These terms will need to be presented on the system interface for the users to choose. Especially for mobile devices, this concept would be too expensive to implement due to the limitations mentioned earlier. Therefore, imprecise queries which freely combined any schema term must be allowed as an alternative. Imprecise query has been the subject of many researchers. It has been widely used especially in information retrieval [5][6]. However, for database, imprecise query was only discussed in some keyword-based schema-less related query systems such

as those of Agrawal, Chaudhuri and Das [7] and Calado et al. [8]. Both works allow users to enter queries using content as search terms, e.g., *John*. Furthermore, these systems were developed for usage on conventional devices which have fixed network connection. As for small mobile devices, to the authors' knowledge, no such querying concept is yet to be found.

Furthermore, the word "unplanned" in database querying is rather subjective. This is because, like any other computer-related operations, there is always a limit to the kind of operations which can be executed. For database, this range of operations depends on the level of expressiveness that a query language exhibits [9]. Chandra [10] mentioned in his work that a language can support relational level of expressiveness at the lowest level to computable expressiveness at the highest end; and these languages can be of different forms. For example SQL is a textual language which exhibits at least relational level of expressiveness (it supports the five basic operations of Selection, Projection, Join, Union and Set Difference) and Query-By-Example is a graphical language which is also capable of supporting relational expressiveness. For mobile devices, Polyviou, Samaras and Evripidou [11] discussed expressive queries in their work which implement directory-like interface for query formulation. However, their method is only suitable for devices that have pen input mechanism. In other words, for users who have cell phones with only the basic input features such as keypad and function keys, the method of expanding and contracting several sub-nodes in the directory will be tedious to carry out. Another interesting option to be considered for inputs on cell phones would be voice input. However, as being highlighted in [12] and [13], this method has one major problem of voice recognition which needs further extensive research.

With the above brief background of related works, the next section will present the free-form query language proposed.

III. FREE-FORM QUERY LANGUAGE

The free-form query language proposed has the structure as shown in Fig. 1. The structure denotes that the language supports the relational level of query expressiveness. This is resulted from an initial data collection exercise conducted to 45 non-expert database users. Non-expert here basically means that the users have had an experience of interacting with a database but still at a very minimal level. This is necessary in order to develop a language as simple as possible that can cater even the novice users. At the same time this group of users was also selected so that they can return meaningful results for the data collection purposes due to their knowledge of what a database is all about. A test database was described to the participants and they were asked to write down at least 5 queries or information that they would want to get from the database. 262 queries were received and from that number, 42 queries (16%) were of the projection type, 86 queries (33%) of the selection type, 121 queries (46%) of the join type, and 5 queries (2%) and 8 queries (3%) were of the union and the set difference types respectively. Reference [14] presents the detail of the above exercise and its results.

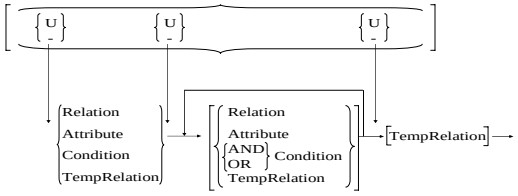

Fig. 1 Structure of free-form query language
(Note: {} denotes either/or, [] denotes optional)

The structure provided in Figure 1 shows, firstly, the terms that can be used to form a query using the language are schema-based terms, i.e., relation names and attribute names of the intended database. The rationale behind this is due to the limitations of cell phones as described earlier. Schema-based terms imply that ambiguities and time needed for matching query terms to the instances or content stored in the database can be reduced. Hence, refinement of the queries which require interactions between cell phones and the database server can be minimized or eliminated altogether. Secondly, the structure also shows that a query term can be used on its own or combined with other terms in any way preferred by users. In other word, this concept of free combination is similar to those adopted by keyword-based languages such as [7] and [8]. However, besides using content as query terms, both [7] and [8] only allow equal comparison as compared to the free-form language proposed which provides avenue for constructing other types of comparison such as larger than, smaller than etc.

Using the structure above, queries such as q1, q2, q3 and q4 below are all valid queries. Queries, q1 combines two relations, while, q2 combines two attributes (not necessarily from the same table). A query which combines a relation and an attribute is also acceptable (see q3) and the order of the terms in the query can be reversible as well. A query, q4, which combines two conditions, is also allowable. (Note: For the purpose of distinguishing between relation names and attributes terms, in this paper the former are capitalized, while the latter are not. In the actual setting, the language is case insensitive)

q1: STUDENT SUBJECT
q2: SUBJECT.subjname STAFF.stafname
q3: SUBJECT STAFF.stafname
q4: STAFF.stafID='e0001'AND SUBJECT.subjcrhr>2

Furthermore, the language also allows a union or a set difference query to be implemented by including the respective operator once, anywhere in the query. However, the position of the set operator determines the components of the query which needs to be manipulated. For example, a query, q5, requests for students to be combined with staff who teaches third year students. On the other hand, a query, q6,

requests for third year students to be combined with staff.

q5: STUDENT.studname U STUDENT.studyear=3 STAFF.stafname
q6: STUDENT.studname STUDENT.studyear=3 U STAFF.stafname

Besides the basic operations described above, the free-form language can also be used to formulate complex operations such as joining the results of a previous union operation with a new relation. This is done by keeping the result of the previous operation in a temporary relation, i.e., denoted as *TempRelation* in Fig. 1. The stored relation can then be used in a successive operation. Below is an example of such operation where the name of all third year students and all staff are combined and saved in a temporary relation called *TEMP1*. Later, the content of *TEMP1* is joined with the *SUBJECT* relation to display the subject(s) that each individual takes.

q7: STUDENT.studname STUDENT.studyear=3 U STAFF.stafname TEMP1
q8: TEMP1 SUBJECT

Hence, using the free-form language, any relational query can be formulated without the need to provide precise inputs and consequently, can minimize the number of query terms required. Especially for queries which involve two distantly related relations, no intermediate links shall be needed by the free-form language. It is enough to provide a term for the needed information only. For example, to query the venues in which a staff gives lectures, it is sufficient to mention only *STAFF* and *VENUE* in the free-form query; even though in actual fact, the relations *STAFF* and *VENUE* might be related via relations of *SUBJECT* and *SESSION*. Furthermore, there is also no need for users to mention the conditions that hold any relationship between any two relations. These relationships are automatically determined by the system based on the path(s) that connects them. Making intermediate links transparent to users would greatly benefit users, since, as stated earlier in the results from the initial data collection exercise, the majority of queries posted by the participants are of the join type.

Since queries in the free-form language are rather vague in terms of specifying the attributes to be presented as outputs, a survey was conducted to capture user expectation of different types of query term when they are used on their own or together with other terms. 13 different queries were presented to 96 survey respondents who were asked to choose the output that they expect from the queries. Table 1 shows the results of the first five questions in the survey which were related to a standalone term.

TABLE I
SINGLE TERM QUERIES AND THE OUTPUT SELECTED

Single term query	Output selected	% respondents selecting
(1) Relation	All attributes of each record in the relation	64.6
(2) Attribute	The stated attribute of each record in the relation	90.6
(3) Condition (primary key)	All attributes for all matched records in the tested relation	63.5
(4) Condition (non-key)	All attributes for all matched records in the tested relation	57.3
(5) Condition (string matching)	All attributes, for all records that contain the string in the attribute tested (inexact match)	65.6

As for multiple terms, Table II shows the output selected by respondents.

TABLE II
MULTIPLE TERMS QUERIES AND THE OUTPUT SELECTED

Multiple terms query	Output selected	% respondents selecting
(6) Relations	Natural join of all attributes of all relations according to order in query	85.40%
(7) Relations + attributes	Natural join of all attributes if term is a relation, or the stated attribute if term is attribute according to order in query	89.60%
(8) Relations + conditions	Natural join of all attributes of all relations that satisfy conditions	53.10%
(9) Attributes + conditions	All stated attributes for which natural join of relations satisfy conditions	51.00%

(10) Conditions	All attributes of the relation in the first condition whose records satisfy all conditions	51.00%
(11) First term set operator	Output of the first term minus output from combination of the rest of the terms	78.10%
(12) Last term set operator	Output of the first term minus output from combination of the rest of the terms	70.80%
(13) Intermediate term set operator	Output from combination of terms before operator minus output from combination of terms after operator	79.20%

In summary, both of the above tables say that a relation name in a query denotes that all attributes of the relation are to be displayed as output unless an attribute of the relation also exists in the query, in which case, only the specified attribute will be displayed. Conditions alone in a query will display all attributes of the relation mentioned in the first condition. This relation is the one stated on the left hand side of the comparison operator. For example, *STUDENT.year=3 SUBJECT.name='TAB1033'* will display all particulars of the third year students who take TAB1033. This rule will be overwritten if there are other query terms which are relation name or attribute name exist in the same query. As for set operation queries, all query terms on the left of the set operator (U or -) will be considered as one query, and all query terms on the right hand side will be another query. However, if the set operator is the first or the last character in the query composed by users, then, the first term will make up the first query and the rests compose the second query.

IV. TRANSLATION TO SQL STATEMENTS

As the free-form language can be considered as high level, an approach needs to be developed for converting or translating the free-from queries into SQL statements. For this, two database relations are used. One of these relations, named as *system$paths*, stores five shortest-most paths between every two relations in the database. Every time a query that involves more than one relation is issued, *system$paths* will be consulted for finding the in-between relations of every adjacent pair of the relations mentioned in the query. These in-between relations together with the query relations are used as items to be included in the FROM clause of the SQL query. The other system relation, *system$metainfo*, contains information on each attribute of each relation in the database. It is referred to when formulating the WHERE clause of a join query. Tuple that contains the primary keys of every pair of adjacent relations identified in the FROM clause is searched in the *system$metainfo* relation. The tuple found provides the necessary attributes that linked the two relations and therefore used for formulating a condition in the WHERE clause. Below is the relation scheme of the *system$paths* relation and Table III shows a sample content.

system$paths(relation1, relation2, path)

TABLE III
SAMPLE CONTENT OF *SYSTEM$PATHS* RELATION

Relation1	Relation2	Path
9	4	3$8$4$

Each relation is given a number to identify them. The 9 in the sample content above denotes relation number 9, the 4 denotes relation number 4, and 3$8$4$ denotes the path between the two relations are relation number 3 followed by relation number 8.

As for the *system$metainfo* relation, below is its relation scheme followed by Table IV with some sample contents.

system$metainfo(columnName, columnType, primaryKey, foreignKey, relationName, baseRelationName, baseAttributeName)

TABLE IV
SAMPLE CONTENTS OF *SYSTEM$METAINFO* RELATION

Col. name	Col. type	P.K.	F.K.	Rel. name	Base rel. name	Base attr. name
Code	char(7)	Y	NULL	subject	NULL	NULL
lecturer	char(5)	NULL	Y	subject	staff	ID
Name	char(30)	NULL	NULL	subject	NULL	NULL

The first sample says that the attribute *code* is a primary key of the *SUBJECT* relation. The second sample denotes that *lecturer* is a foreign key attribute in the *SUBJECT* relation. It refers to the *ID* attribute in the *STAFF* relation. The third sample on the other hand, is showing that *name* is an attribute in the *SUBJECT* relation. It is neither a primary key nor a foreign key attribute.

Fig. 2, Fig. 3, Fig. 4, Fig. 5 and Fig. 6 are flowcharts that describe the translation process of a free form query to its SQL statement. To demonstrate how the process works, let's translate the query below:

STAFF.name STUDENT.name SUBJECT

The query will be first checked for set operator as shown in the flowchart of Fig. 2. Since it does not have any set operator, the *Evaluate expression* module will be invoked.

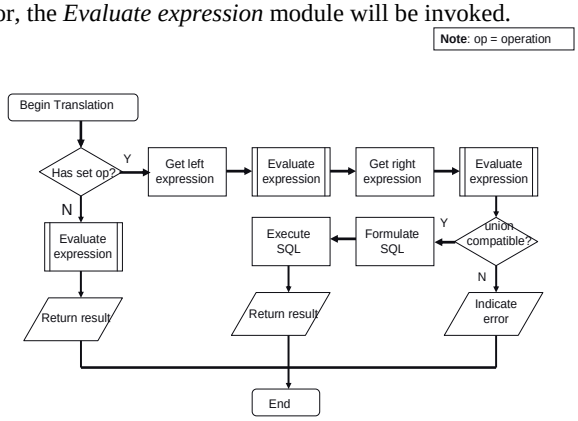

Fig. 2 Flowchart for translation to SQL statement

Evaluate expression module as presented in Fig. 3 will evaluate the type of every query term in the query. Both *STAFF.name* and *STUDENT.name* are attributes and therefore, they will be kept in the *col array*. Their relations will also be stored in the *tab array* as well. As for the *SUBJECT* relation, it will be kept in the *tab array* without any attribute in the *col array*. After all terms have been evaluated, the *Form SELECT clause* module of Fig. 4 will be invoked.

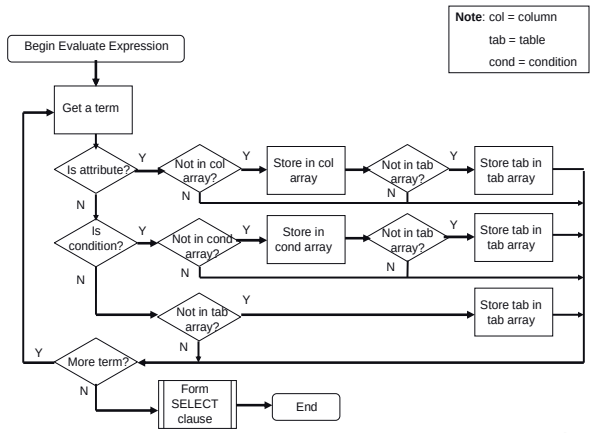

Fig. 3 Flowchart for Evaluate expression module

At the *Form SELECT clause* module as shown in Fig. 4, each relation in the *tab array* will be checked to see if it has any attribute in the *col array*. If there is none, the whole attributes of the relation will be included in the SELECT clause, otherwise, the attributes in the *col array* will be appended to the SELECT clause. For the sample query, both *STAFF.name* and *STUDENT.name* will be included in the SELECT clause. However, for *SUBJECT*, all of its attributes will be appended to the SELECT clause.

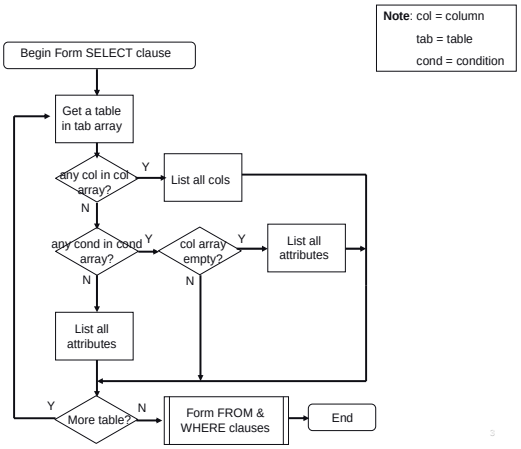

Fig. 4 Flowchart for Form SELECT clause module

After the formulation of the SELECT clause, the *Form FROM & WHERE clauses* module as in Fig. 5 will be next executed. To make things simple, let's assume there is only one path between the relations *STAFF* and *STUDENT* and one path between *STUDENT* and *SUBJECT*. Hence, the FROM clause will include all relations involved, either direct or indirect, and the WHERE clause will have all links between these relations.

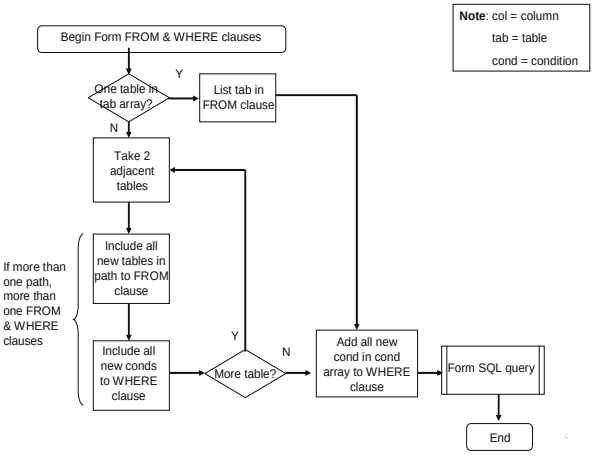

Fig. 5 Flowchart for Form FROM & WHERE clauses module

After the FROM and WHERE clauses have been formulated, the three clauses will then be passed to the *Form SQL query* module for integration. Fig. 6 shows this last process of formulating the SQL statement.

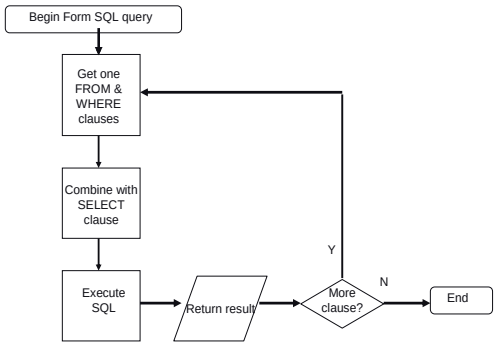

Fig. 6 Flowchart for Form SQL query module

The result of all the above processes ends with the following SQL statement which is next passed to the database server for execution.

```
SELECT STAFF.name STUDENT.name SUBJECT.code,
SUBJECT.name, SUBJECT.lecturer
FROM STAFF, STUDENT, SUBJECT, ENROLMENT
WHERE STAFF.ID=SUBJECT.lecturer AND
SUBJECT.code=ENROLMENT.subjCode AND
ENROLMENT.studentID=STUDENT.ID
```

V. CONCLUSION

As a conclusion, it is possible to develop a database query formulation system for mobile phones which accepts unplanned queries, by allowing free-form inputs. Since cell phones are poor in terms of resources as compared to other mobile devices, the successfulness of implementing such a method on them would mean it is applicable to the other devices. Imprecise query method in the form of free-form language can provide a much simpler interface for users to formulate queries. The method can also help in reducing the number of query input especially in cases where joins of relations are needed. Since the majority of queries which might be issued are of this type (as seen by the queries given by respondents), providing such a method would benefit users of resource-poor devices. For the future work, more complex queries such as those involving aggregate functions, grouping etc. will be implemented and so are other types of database transaction such as delete, insert and update.

REFERENCES

[1] R. Alonso, and H. F. Korth, "Database system issues in nomadic computing", in *Proc. 1993 SIGMOD Conference*, Washington D.C., 1993, pp. 388-392.

[2] K. Hung, and Y-T. Zhang, "Implementation of a WAP-based telemedicine system for patient monitoring," *IEEE Transactions on Information Technology in Biomedicine*, Vol. 7, No. 2, June 2003, pp. 101-107.

[3] A. Koyama, N. Takayama, L. Barolli, Z. Cheng, and N. Kamibayashi, "An agent based campus information providing system for cellular phone," in *Proc. 1st International Symposium on Cyber Worlds*, Tokyo, 2002, pp. 339-345.

[4] P. Boonsrimuang, H. Kobayashi, and T. Paungma, "Mobile Internet navigation system," in *Proc. 5th IEEE International Conference on High Speed Networks and Multimedia Communications*, Jeju Island, 2002, pp. 325-328.

[5] A. Bergstrom, P. Jaksetic, and P. Nordin, "Enhancing information retrieval by automatic acquisition of textual relations using Genetic programming," in *Proc IUI 2000*, New Orlean, 2000, pp. 29-32.

[6] H-M. Lee, S-K. Lin, and C-W. Huang, "Interactive query expansion based on fuzzy association thesaurus for web information retrieval, " in *Proc. IEEE International Fuzzy Systems Conference*, Melbourne, 2001, pp. 724-727.

[7] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases," in *Proc. IEEE 18th International Conference on Data Engineering (ICDE'02)*, San Jose, 2002, pp. 5-16.

[8] P. Calado, A.S. da Silva, A.H.F. Laender, B.A. Ribeiro-Neto, and R.C. Viera, "A Bayesian network approach to searching web databases through keyword-based queries, " *Information Processing and Management*, Vol. 40, No. 5, September 2004, pp. 773-790.

[9] R. Ramakrishnan, J. Gehrke, *Database Management Systems*. McGraw-Hill, New York, 2000.

[10] A. Chandra, "Theory of database queries," in *Proc. 7th ACM Symposium on Principles of Database Systems*, Texas, USA, 1988, pp. 1-9.

[11] S. Polyviou, G. Samaras, and P. Evripidou, "A relationally complete visual query language for heterogeneous data sources and pervasive querying," in *Proc. 21st International Conference on Data Engineering (ICDE 2005)*, Tokyo, 2005, pp. 471-482.

[12] E. Chang, F. Seide, H.M. Meng, Z. Chen, Y. Shi, and Y.C. Li, "A system for spoken query information retrieval on mobile devices," *IEEE Transactions on Speech and Audio Processing*, Vol. 10, No. 8, November 2002, pp. 531-541.

[13] B. R. Bai, C.L. Chen, L.F. Chien, and L.S. Lee, "Intelligent retrieval of dynamic networked information from mobile terminals using spoken natural language queries," *IEEE Transactions on Consumer Electronics*, Vol. 44, No. 1, February 1998, pp. 62-72.

[14] R. Ahmad, S. Abdul-Kareem, "A free-form database query language for mobile phones," in *Proc. International Conference on Communications and Mobile Computing*, Kunming, 2009, pp. 279-284.