Chameleon: Context-Awareness inside DBMSs*

Hicham G. Elmongui #1, Walid G. Aref #2, Mohamed F. Mokbel ‡3

#*Department of Computer Science, Purdue University*
305 N. University St., West Lafayette, IN 47907, USA
[1]elmongui@cs.purdue.edu
[2]aref@cs.purdue.edu

‡*Department of Computer Science and Engineering, University of Minnesota*
200 Union Street SE, Minneapolis, MN 55455, USA
[3]mokbel@cs.umn.edu

***Abstract*— Context is any information used to characterize the situation of an entity. Examples of contexts includetime, location, identity, and activity of a user. This paper proposes a general context-aware DBMS, named Chameleon, that will eliminate the need for having specialized database engines, e.g., spatial DBMS, temporal DBMS, and Hippocratic DBMS, since space, time, and identity can be treated as contexts in the general context-aware DBMS. In Chameleon, we can combine multiple contexts into more complex ones using the proposed context composition, e.g., a Hippocratic DBMS that also provides spatio-temporal and location contextual services. As a proof of concept, we construct two case studies using the same context-aware DBMS platform within Chameleon. One treats identity as a context to realize a privacy-aware (Hippocratic) database server, while the other treats space as a context to realize a spatial database server using the same proposed constructs and interfaces of Chameleon.**

I. Introduction

According to the Merriam-Webster Online Dictionary, the term "context" is defined as *the interrelated conditions in which something exists or occurs* [1]. Many researchers have tried to define context. However, often definitions use examples of context (e.g. [2]) or synonyms [3], [4]. In the Ubiquitous Computing community, context is defined as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [5]. Examples of contexts include, but are not limited to, time, location, identity, and activity of a user.

Context-aware computing (introduced in [2]) describes a system to be context-aware if "it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task" [5]. Context-aware personalization is about tailoring services to better adapt to user preferences with knowledge about the user and his/her environment.

In this paper, we introduce *context-awareness inside DBMSs*. In contrast to all existing work about context aware systems that are built on top of an infrastructure that provides just the data (e.g., database or data stream), we propose to incorporate context awareness inside the DBMS. Several specialized DBMSs, which manage data and answer queries related to one context type, already exist. For instance, a spatial DBMS is optimized to manage and query objects in space. A temporal DBMS has a built-in temporal data model. Tailoring such database engines is not an easy task. Moreover, supporting and combining multiple contexts in one tailored database engine is not within reach. By including the concept of contexts inside the DBMS, we will not need to tailor specialized engines towards a certain context (let alone multiple contexts), but rather we will be able to have systems that support user-defined complex and composite contexts.

We present the design and implementation of Chameleon, a Context-Aware DBMS. Chameleon[1] supports multiple contexts as well as user preferences and has a generic interface to define and process context information. Contexts in Chameleon are classified according to their properties. As a proof of concept, Chameleon is used to support basic contexts, such as location and identity and show how to compose complex contexts from basic ones.

II. Classification of Contexts

Context type	object context user context	**Contextual relation**	[Q] equivalence relation [T] total ordering relation [P] partial ordering relation
Context sign	[S] positive [G] negative	**Listing of Data**	[X] unlisted excluded [N] unlisted included

Two main entities are involved in a context-aware database system: the query issuer and the data being queried. Both of these entities may have their own contexts.

A. Object Context

Object context is the context of the data being queried. Object context might be one or more object attributes. Otherwise, the object context might need to be defined, and instantiated apart from the object data.

B. User Context

User context refers to the context of the query issuer. It can be location, identity, preferences, or any information relevant to the user. When an object context conforms with the user

*This work is supported in part by the National Science Foundation under Grant Numbers IIS-0811954, IIS-0811998, IIS-0811935, and CNS-0708604. Elmongui is also affiliated with Alexandria University, Alexandria, Egypt.

[1]The *real* Chameleons also change their color and appearance based on the context they are in.

context, the object is returned when its table is queried. We classify user contexts according to three dimensions: context sign, contextual relation, and listing of data.

Dimension 1: Context Sign: A user context can be "positive" or "negative". A positive context defines what the context *is*, while a negative context defines what the context *is not*.

Dimension 2: Contextual Relation: Contextual relation is the relation among contextual data. It shows the order of relevance of contextual data. Contextual relation can be an *equivalence*, a *total ordering* or a *partial ordering* relation.

Dimension 3: Listing of Data: Listing of data refers to how the data should be listed; whether out of context data should be excluded from the listing or come last. This is termed as "unlisted excluded" or "unlisted included" respectively.

C. User Context as a 3D Point

Each user context is looked upon as a point in the 3D space defined above. For instance, a user might be equally interested in seeing a horror or a humor movie. This is an example of a positive user context having an equivalence contextual relation with the unlisted contextual values excluded.

All points in this 3D space are valid when the user context is positive. However, when the user context is negative, only contextual values with the unlisted included and equivalence contextual relation are valid.

III. Context-Awareness SQL Constructs

In this section, we describe the different constructs that are needed to enable context awareness inside a DBMS. Chameleon provides support for all these constructs.

Creating Object Contexts: Chameleon uses the `CREATE OBJECT CONTEXT` statement to define an object context when the object context is not part of the object relation.

```
CREATE OBJECT CONTEXT contextname (
        {col_spec | table_constraint} [, ...]
        , table_binding
    );
```

Similar to the `CREATE TABLE` statement, col_spec refers to the specification of an attribute of the object context such as name, data type, and so on. On a similar vein, table_constraint refers to any constraints on the whole context such as check constraints. The construct table_binding is the construct that connects the object with its context. It has the format below.

```
BINDING KEY ([col_name [, ...]])
    REFERENCES ref_table [( ref_col [, ...] )] WITH bool_expr
```

The first part of the `BINDING KEY` is similar to the `FOREIGN KEY`. However, the binding key does not have to reference a primary key. Moreover, the decision to bind a contextual value with an object does not have to be an equality with a column value in the referenced table. The `WITH` construct defines a Boolean expression that serves as the binder. Furthermore, the binding key might not contain any context attribute, but rather only the Boolean expression that might also contain attributes from any object context to the referenced table.

Creating User Contexts: Similar to object contexts, each user context will materialize to a relation. Chameleon uses the following syntax to define a user context. For each table affected by a user context, a binding key is used to show how the context reflects on the table. Therefore, there might be more than one binding key in a user context.

```
CREATE [context_sign] CONTEXT contextname (
        {col_spec | table_constraint} [, ...]
        , table_binding [, ...]
        [, substituting_key [, ...]]
    )
    [AS contextual_relation_clause]
    [WITH UNLISTED unlisted_status];
context_sign: positive | negative
contextual_relation: equivalence | partial order
                | total order [USING ordering_func]
unlisted_status: excluded | included
```

Upon creation of a user context, an implicit column is created to hold the *user_name* of the current user. Therefore, each contextual value is associated with a certain user. Also, if an ordering relation is used for the contextual relation, then another implicit column is created to hold the *rank* of that contextual value. This rank can be either input by the application while acquiring contextual data, or it can be computed using an ordering function *ordering_func*. In the latter case, the rank column does not need to exist.

The substituting_key will be discussed in details in the next construct. Populating the contextual relations is made using standard SQL `INSERT` statements. Also, other data manipulation statements will still work on the contextual relations.

Global Substitution Construct: Some attributes need to be modified for presentation purposes if we want to enable context awareness. For instance, if the context is the location of a user, and the user is currently in France, then we might want all prices, in all tables, to be converted to Euro. This conversion is called *global substitution*, since the substitution occurs for all tables according to the current context. The substituting key defines such conversion, and is defined while defining the user context as follows.

```
SUBSTITUTE table_name(col_name) BY expression;
```

The expression that substitutes the attribute can be any expression in which attributes from table_name, its object contexts, as well as the user context may appear.

Setting Active Contexts: The application user may have many contexts, not all of them need to be current all the time. Therefore, we introduce the construct `SET ACTIVE CONTEXT` to define the current contexts to be taken into account for that user. The user_name has the CURRENT_USER as a default.

```
SET ACTIVE CONTEXT [FOR USER user_name]
AS context_name [, ...];
{ [WITH RANKING ORDER context_name [, ...] ]
  | [WITH RANKING EXPRESSION expression
  | [WITH SKYLINE OF expression {MAX|MIN} [, ...] ] };
```

The `SET ACTIVE CONTEXT` context provides for composing complex contexts from basic ones. If more than one context imposes an ordering on the data, we provide for three mechanisms for the composed ordering: (1) by specifying a context order upon which the data is to be sorted, (2) by specifying a ranking expression to be used if a ranking algorithm is to be used (e.g., [6]), and (3) by requesting the skyline of the data when the ranks of different contexts are inversely correlated.

Chameleon provides for a SkylineJoin query operator, which not only provides the skyline of the data but also seizes the opportunity that the ranks come from different tables to produce the results fast. The details of this operator are out of the scope of this paper and may be found in our technical report [7].

IV. Conceptual Evaluation

We use a preference-based system to show why the above constructs enable context-aware query processing. Consider a table books(id, title, year, category, cover, stock) that contains information about books in a certain bookstore.

Example contexts are given and defined using the above constructs. We start by simple contexts then combine them into more complex ones. In the scenarios below, the user executes the following query (Q_u), and the relevant tuples are returned.

```
SELECT * FROM books WHERE books.stock;
```

Context 1: The user has a preference for only books of a certain category (e.g., Science fiction).
This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_category_SQX (
        category varchar(20),
        BINDING KEY (category) REFERENCES books(category)
    ) AS EQUIVALENCE WITH UNLISTED EXCLUDED;
SET ACTIVE CONTEXT AS ctxt_category_SQX;
```

We give the suffix `SQX` to the context name above to emphasize that it is a positive `[S]` context with an equivalence `[Q]` contextual relation and that the unlisted categories in the context are to be excluded `[X]` . For the above example, when the user issues Q_u, the binding key is used to join the books table with the context table, and only the books whose category exists in the context are to be returned.

Context 2: The user's preference is for books published in 2005, and then those published in 2006 before all other books.
This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_year_STI (
        year integer,
        BINDING KEY (year) REFERENCES books(year)
    ) AS TOTAL ORDER WITH UNLISTED INCLUDED;
SET ACTIVE CONTEXT AS ctxt_year_STI;
```

For the above example, in response to Q_u, the binding key is used to join the books table with the context table. In this case, the type of join is a left outer join, and therefore, all books will be returned at the end. The output rows are to be sorted based on the year *rank*, which is specified implicitly in the context as it is an ordering context. Rows with NULL context rank appear later in the list.

Context 3: User prefers hardcover to paperback books.
This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_cover_STX (
        cover integer,
        BINDING KEY (cover) REFERENCES books(cover)
    ) AS TOTAL ORDER WITH UNLISTED EXCLUDED;
SET ACTIVE CONTEXT AS ctxt_cover_STX;
```

For the above example, in response to Q_u, the binding key is used to join the books table with the context table. The output rows are to be sorted based on the cover *rank*, which is specified implicitly in the context as it is an ordering context.

Context 4: User does not prefer science fiction books.
This context may be defined as:

```
CREATE NEGATIVE CONTEXT ctxt_category_GQI (
        category integer,
        BINDING KEY (category) REFERENCES books(category)
    ) AS EQUIVALENCE WITH UNLISTED INCLUDED;
SET ACTIVE CONTEXT AS ctxt_category_GQI;
```

In response to Q_u, rows in books, whose category exists as any of the contextual values of this context, are eliminated from the answer set.

Next, we compose complex contexts from the above basic contexts. We start with the following context.

Context 5: User prefers books published in 2005, and then those published in 2006 before all other books. For the books that are similarly ranked, the user prefers hardcover books over books with paperback cover.
This context may be viewed as the composition of ctxt_year_STI and ctxt_cover_STX. We just need to set the active context appropriately to reflect to the desired context.

```
SET ACTIVE CONTEXT FOR user1
    AS ctxt_year_STI, ctxt_cover_STX
    WITH RANKING ORDER ctxt_year_STI, ctxt_cover_STX;
```

As a result of this combined context, queries to select tuples from books will work as follows. First, the books in stock will be sorted based on the rank of the years, and then in case of ties, the cover type will be taken into consideration.

V. Instantiating Hippocratic Databases

We show how to limit disclosure, as in Hippocratic Databases, using context awareness in Chameleon. Table I shows the same patient table used in the limiting disclosure work in [8]. This table contains patient personal information.

pid	name	age	address	phone
1	Alice Adams	10	1 April Ave.	111-1111
2	Bob Blaney	20	2 Brooks Blvd.	222-2222
3	Carl Carson	30	3 Cricket Ct.	333-3333
4	David Daniels	40	4 Dogwood Dr.	444-4444

TABLE I
THE PATIENT TABLE

Consider a healthcare facility where admitted patients sign a privacy policy that specifies which information can be disclosed to which recipient and for what purposes. On an opt-in basis, the healthcare facility also allows patients to choose if they want any of their personal information to be released to other recipients. For instance, patient's name, age and phone number is disclosed to the treating nurse; while patient's address is not. The patient may opt-in and choose that only her age is to be released to charity for solicitation.

Beside limited disclosure, limited retention is also modeled using context awareness. For simplicity, and without loss of generality, we assume that patient data is retained for 90 days.

Note that objects are patients *in this context*. Object contexts are the contexts of the patients, and users are those who retrieve patients' data at the facility. To model the above example of limiting the disclosure and retention of patients' data in Chameleon, we define the object contexts patient_privacy_pref and patient_policy_signature as in Table II.

Let the object context patient_privacy_pref contain the contextual data in Table III. The user context below enforces limited disclosure and limited retention of patients' data. Table IV shows the data retrieved upon executing the query

```
CREATE OBJECT CONTEXT patient_privacy_pref (
        recipient varchar(30),     purpose varchar(30),     pid integer,
        pid_pref boolean,          name_pref boolean,       age_pref boolean,
        address_pref boolean,      phone_pref boolean,
        BINDING KEY(pid) REFERENCES patient(pid)
    );
CREATE OBJECT CONTEXT policy_signature (
        pid integer,       sign_date date,
        BINDING KEY(pid) REFERENCES patient(pid)
    );
CREATE POSITIVE CONTEXT identity_activity (
        job varchar(30),         activity varchar(30),
       BINDING KEY(job, activity) REFERENCES
          patient_privacy_pref(recipient, purpose)
       SUBSTITUTE patient(pid)
          WITH (CASE WHEN patient_privacy_pref.pid_pref
             AND today() <= policy_signature.sign_date + 90
             THEN patient.pid ELSE NULL)
       SUBSTITUTE patient(name)
          WITH (CASE WHEN patient_privacy_pref.name_pref
             AND today() <= policy_signature.sign_date + 90
             THEN patient.name ELSE NULL)
        .
        .
        .
    ) AS EQUIVALENCE WITH UNLISTED EXCLUDED;
```

TABLE II

INSTANTIATING HIPPOCRATIC DATABASES

recipient	purpose	pid	pid_pref	name_pref	age_pref	address_pref	phone_pref
charity	solicitation	1	√	√	√	√	√
nurse	treatment	1	√	√	√	×	√
charity	solicitation	2	×	×	×	×	×
nurse	treatment	2	√	√	√	×	√
charity	solicitation	3	√	×	×	√	√
nurse	treatment	3	√	√	√	×	√
charity	solicitation	4	√	√	×	×	×
nurse	treatment	4	√	√	√	×	√

TABLE III

THE PATIENT_PRIVACY_PREF OBJECT CONTEXT

(job, activity)	pid	name	age	address	phone
(charity, solicitation)	1	Alice Adams	10	1 April Ave.	111-1111
	3			3 Cricket Ct.	333-3333
	4	David Daniels			
(nurse, treatment)	1	Alice Adams	10		111-1111
	2	Bob Blaney	20		222-2222
	3	Carl Carson	30		333-3333
	4	David Daniels	40		444-4444

TABLE IV

RESULT OF "SELECT * FROM PATIENT;"

"`SELECT * FROM patient;`" by different (job, activity) pairs when the context `identity_activity` is active.

VI. INSTANTIATING SPATIAL DATABASES

Spatial databases are optimized to store and query data about objects in space. This type of databases has more complex geometrical data types, e.g., points, lines, and rectangles.

Consider a real-estate database containing information about houses. The *houses* table has the following schema: houses(id, bedrooms, price, city). An application developer is interested in providing some spatial queries to this database, but has no privileges to add the location of the house to this table. An object context is created to add the location of houses.

Range Queries: Let the user context be the willingness to buy a house in certain regions. As a result, a user context is created in Chameleon to declare that only houses contained in relevant regions are to be returned. These contexts are defined as `house_loc` and `houses_in_region` (see Table V).

```
CREATE OBJECT CONTEXT house_loc (
        id integer,     x integer,     y integer,
        PRIMARY KEY(id),
        BINDING KEY id REFERENCES houses(id)
    );
CREATE POSITIVE CONTEXT houses_in_region (
        x1 integer,    y1 integer,    x2 integer,    y2 integer,
        BINDING KEY() REFERENCES house_loc
              WITH contained(house_loc.x, house_loc.y, x1, y1, x2, y2)
    ) AS EQUIVALENCE WITH UNLISTED EXCLUDED;
CREATE POSITIVE CONTEXT nearby_houses (
        x integer,     y integer,
        BINDING KEY() REFERENCES house_loc WITH true
    ) AS TOTAL ORDER USING dist(x, y, house_loc.x, house_loc.y)
    WITH UNLISTED EXCLUDED;
SET ACTIVE CONTEXT FOR user2
    AS houses_in_region, nearby_houses
    WITH SKYLINE OF nearby_houses.rank MIN, houses.price MIN;
```

TABLE V

INSTANTIATING SPATIAL DATABASES

Nearest Neighbor Queries: Another class of spatial queries is the nearest neighbor query, where a user wants to retrieve the object that is nearest to a pivot location. In the real estate database, a user willing to retrieve the houses listed by proximity to a point may declare her context as `nearby_houses`.

Skyline Queries: Skyline queries emerge in spatial databases. Assume *user2* wants to buy a house that is both near his work and cheap. Since it is not easy to combine such preferences in a ranking expression, user2 decides to select from the skyline houses. Such context can be defined as the composition of `houses_in_region`, `nearby_houses`, and the context *price* already incorporated in the houses table.

VII. CONCLUSIONS

We propose incorporating context awareness inside DBMSs. Not only does this avoid the need to tailor specialized systems for a certain context, but it also allows for combining several contexts to form a complex one. We introduce Chameleon, a prototype context-aware DBMS built by extending PostgreSQL. We provide SQL constructs to enable context awareness, and we present the conceptual evaluation to show how to use the system to define contexts. We include two case studies: Hippocratic databases and Spatial databases as a proof of concept in using Chameleon to instantiate specialized systems.

REFERENCES

[1] "Merriam-Webster Online Dictionary," http://www.m-w.com/.

[2] B. N. Schilit and M. M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, vol. 8, no. 5, 1994.

[3] P. Brown, "The Stick-e document: a framework for creating context-aware applications," *Electronic Publishing*, vol. 8, no. 2&3, 1996.

[4] T. Rodden, K. Chervest, N. Davies, and A. Dix, "Exploiting Context in HCI design for Mobile Systems," in *HCI*, 1998.

[5] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *Workshop on the What, Who, Where, When, and How of Context-Awareness, CHI*, 2000.

[6] I. F. Ilyas, W. G. Aref, A. K. Elmagarmid, H. G. Elmongui, R. Shah, and J. S. Vitter, "Adaptive Rank-Aware Query Optimization in Relational Databases," *TODS*, vol. 31, no. 4, 2006.

[7] H. G. Elmongui, W. G. Aref, and M. F. Mokbel, "Chameleon: Context-Awareness inside DBMSs," Purdue University, Tech. Rep. CSD TR 08-028, 2008.

[8] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in Hippocratic databases," in *VLDB*, 2004.