# Context-aware databases
## *design, integration and applications*

*Letizia Tanca*
*Politecnico di Milano (*)*
*tanca@elet.polimi.it*

*(*) joint work with the Context-ADDICT team:*
*C. Bolchini, C. A. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber*

Bertinoro, March 2008

# *Context in Information Systems*

Context-aware databases

*design, integration and applications*

Bertinoro, March 2008

# *Context as abstraction*

- **abstraction mechanisms are known to be** *indispensable means to deal with complexity* **(see abstractions for database design).**
- **the** *viewpoint abstraction* **has received little attention**
- **context as a** *viewpoint mechanism* **that takes into account implicit background knowledge**

# Categories

- **Context as a selector of *workspaces***

  **→ information-system oriented**

  **→ the first 2 models**

- **Context as a selector of *views* or *facets***

  **→ database oriented**

  **→ the last 3 models (plus ours)**

# Modeling Context and its Applications

(Motschnig-Pitrik, Mylopoulos)

- **Decomposition of an information base into possibly overlapping subsets, referred to as *contexts***
- **Objects, complex or elementary, are called *information units***
- **Mechanisms for partitioning and coping with a fragmented information base have appeared in different forms:**
  - **database views**
  - **multidatabases**
  - **(software, CAD) versions**
  - **workspaces**
  - **knowledge base partitions and contexts**
  - **programming language scopes and scope rules**
  - **hypertext perspectives**

# Modeling Context and its Applications

(Motschnig-Pitrik, Mylopoulos)

**A conceptual, <u>uniform</u> framework for contexts, supporting:**

- **context-specific naming and representation of conceptual entities**
- **relativized transaction execution**
- **operations for context construction and manipulation**
- **authorization**
- **change propagation**

**A context, in the first place, is characterized by** *its contents*

# Modeling Context: example

(Motschnig-Pitrik, Mylopoulos)

**Enterprise Information Base**

- **employee with name Chryss M**
- **in the** *R&D context* **Chryss M. characterized by an object called chryss having the attributes: socialInsuranceNo, name, and projects**
- **in the** *accounting context*, **Chryss M. described by an object having the name chris and the attributes: socialInsuranceNo, dateOfBirth, salary.**
- **The information unit (in OO terms, object) referring to the person Chryss M. gives rise to two unit versions**
- **in OO models, a unit version is referred to as** *perspective* **(or** *perspective object***)**

# Modeling Context: example

(Motschnig-Pitrik, Mylopoulos)

```
unit3 wrt r&d:
identifier is 'chryss'

representation is
   object with attribute

    socialInsuranceNo:
   123456
    name: Chryss M.
    projects: CSCW11, UID2
       …  …
       …  …
    end chryss wrt c1
```

```
unit3 wrt accounting:
identifier is 'chris'

representation is
   object with attribute

   socialInsuranceNo:
   123456
   dateOfBirth: 15/12/60
   salary: 50.000
   …   …
   end chris wrt c2
```

# Modeling Context

(Motschnig-Pitrik, Mylopoulos)

- *Contexts* **are special** *information units*
- **The definition of each context includes four components:**
  - **contents of the context,**
  - **the local names (***lexicon***) used for units within the context**
  - **the authorization rules based on combinations of different users and transactions**
  - **the change propagation links, specified in terms of contexts which shall receive changes from or propagate changes to the context under definition**
- *Identifiers* **are special units consisting of character sequences, distinguished by quotes. E.g., `'john'` denotes a unique four-character string.**

# Modeling Context: selector functions

(Motschnig-Pitrik, Mylopoulos)

- **The selector function *contents* takes as argument a context and returns the set of units included in that context**

```
contents(r&d) = {'Employee','tom','chryss','john',
 'r&d' ,unit1, unit2, unit3, unit4, unit10}
```

- **the selector function *lexicon* maps each context to its lexicon of identifiers and their referents with respect to that context**

```
lexicon(r&d) =

{['Employee',unit1],['tom',unit2],['chryss',unit3],
 ['john',unit4],['r&d',unit10]}
```

# Modeling Context – selector functions

(Motschnig-Pitrik, Mylopoulos)

- *authorP* **selects for each context c and user-transaction pair `(u,t)` a predicate which determines whether user u is authorized to execute transaction `t` within context c**
- *propagateFrom* **indicates those contexts, whose changes shall be received in the context under definition**
- *propagateTo* **indicates which contexts shall be sent changes from the context under definition**

```
authorP(u,t) wrt r&d = authorP(u,t) wrt accounting =
(MakeEmployee(t) ∧ ExpertUser(u)) ∨
(DeleteEmployee(t) ∧ ExpertUser(u)) ∨
(UpdateEmployee(t) ∧ User(u))
propagateFrom(c,u,t) = accounting(c) wrt r&d
propagateTo(c,u,t) = accounting(c) wrt r&d
propagateFrom(c,u,t) = r&d(c) wrt accounting
propagateTo(c,u,t) = r&d(c) wrt accounting
```

# Modeling Context – selector functions

(Motschnig-Pitrik, Mylopoulos)

- *propagateFrom* **indicates those contexts, whose changes shall be received in the context under definition**
- *propagateTo* **indicates which contexts shall be sent changes from the context under definition**

**The need for change propagation arises in all applications where contexts have a non-empty intersection and want to communicate over "shared" units, or want to keep units in their intersection in identical versions, thus realizing a common interface**

# Modeling Context

## (Motschnig-Pitrik, Mylopoulos)

- **An information base (IB) is a collection of contexts**
- **since contexts are units, they could be contained in other contexts**
- **containment may be recursive**
- **although only a single version of some unit may be visible within one context at one point in time, nested contexts may contain further versions of that unit**
- **each context is assigned one or more owners**
- **an owner is authorized to perform any operation or transaction on his/her context, including an operation that constrains this unrestricted authorization**
- *constrainOwner(pred)* **serves as a security mechanism for the owner, and sets an owner's access rights and can only be executed by the owner with respect to the owned context**

# Operations on Contexts

## (Motschnig-Pitrik, Mylopoulos)

- **operations that create new contexts from sets of units or in terms of existing contexts**
  - **it is assumed that each newly created context is added to the contents of the context with respect to which the current transaction is executed**
- **operations for manipulating the contents, lexicon, authorization predicate, and change propagation specification of existing contexts**

**Operations will be discussed in detail with the next model  which extends and further defines this one**

# Operations on Contexts: examples

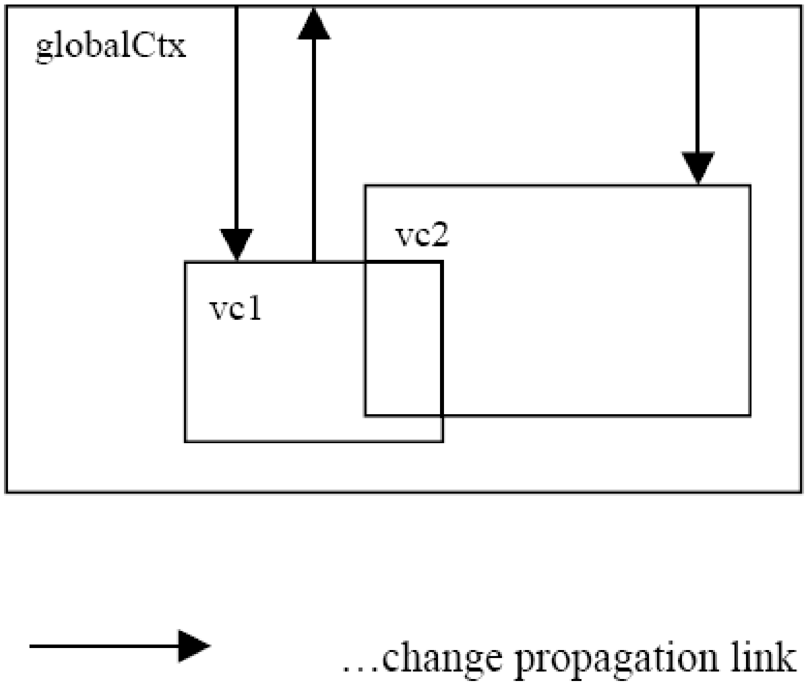
(Motschnig-Pitrik, Mylopoulos)

```
c3 :=newContext({Employee wrt r&d, 'Employee'}, /*contents*/
      {['Employee', Employee wrt r&d]},         /*lexicon*/
       (Query(t)∧ Manager(u),                   /*authorization*/
        false, accounting(c))          /*change propagation*/
```

Creates a new context `c3`, adds it to the contents of the `accounting` context and adds an entry to the accounting context's lexicon. `c3` contains two units: `Employee wrt r&d` and its external identifier `Employee`, in addition to `null` and to itself along with its name `(c3,'c3')`. In `c3`, the name `Employee` is associated with the unit `Employee wrt r&d` and the name `c3` is implicitly associated with some internal unit identifier. The authorization predicate only allows `Query` transactions to be executed by `Manager` users and specifies that no transactions effected in other contexts shall be considered in the newly created context and that all transactions effected in `c3` shall be propagated to the `accounting` context.

# Context-based design template example

## (Motschnig-Pitrik, Mylopoulos)




globalCtx: authorP(u,t) = true
propagateFrom(u,t) = vc1(c)
progagateTo(u,t) = c $\in$ {vc1,vc2}

vc1: authorP(u,t) = true
propagateFrom(u,t) = globalCtx(c)
propagateTo(u,t) = globalCtx(c)

vc2: authorP(u,t) = Query(t)
authorP(owner,t) = Query(t)
propagateFrom(u,t) = globalCtx(c)
propagateTo(u,t) = false

**the use of contexts for modeling views in database systems**

# Modeling Context and its Applications

(Motschnig-Pitrik, Mylopoulos)

- **the proposal does not build on the assumption that a database is defined in terms of a single, global schema:**
  - **an information base can contain one schema per context and allows overlapping**
- **due to relativization, contradictory extensions of the schemata may co-exist within a single information base**
- **the model of change propagation extends the standard model of change propagation associated with views**
- **operations for context creation and extension are provided**
- **capability of defining a context as the result of a query**

# Context in Information bases
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

- Context is used for partitioning an information base into manageable fragments of related objects
- Contexts are complex information objects, associated with a set of objects and a lexicon
- Object names are not unique (differently from the approach of Mylopoulos and Motschnig-Pitrik). Name conflicts are solved:
  - Synonyms
  - Homonyms
  - Anonymous objects
- Objects are referred to contexts
- Context-manipulation primitives

# Definitions

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

- **A** *context* **is a higher order conceptual entity that describes a group of conceptual entities from a particular standpoint**
- **They reflect real-world environments**
- **Nesting of contexts is allowed**
- **A lexicon** $l$ **is a set of pairs** $o{:}l(o)$
- **For a context** $c$**, containing objects** $\{o_1, \ldots o_k\}$,

$$lex(c) = \{o_1{:}N_1, \ldots o_k{:}N_k\}$$

# Example

(Theodorakis, Analyti, Constantopoulos, Spyratos)

- ***lex($c_1$)= { $o_1$:prof.Tanca,***
  ***$o_5$:professor,***
  ***$c_2$:Databases,***
  ***$c_3$:InformationSystems}***
- ***lex($c_2$)= { $o_1$:Letizia,***
  ***$o_2$:head,***
  ***$o_3$:Cristiana,***
  ***$o_4$:Fabio, FabioAlberto}***
- ***lex($c_3$)= {$o_6$:Letizia,***
  ***$o_2$:head,***
  ***$o_1$:Tanca}***

# Definitions
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

- (object reference) For all ***c***, ***o*** recursively contained in ***c***, ***refs(o,c)*** is the set of all names of ***o*** in ***c*** and in all subcontexts of ***c***. Let ***DEI*** be the name for context $c_1$
- $o_1$ can be called:
  - ***prof.Tanca***: indeed $refs(o_1, c_1)$=*{prof.Tanca, Letizia,Tanca}*,
  - ***Databases.Letizia***: indeed $refs(o_1, c_2)$=*{Letizia}*,
  - ***InformationSystems.Tanca:*** indeed $refs(o_1\ c_3)$=*{Tanca}*,
- $o_1$ and $o_6$ have the same name in two different contexts: $refs(o_1, c_2) = refs(o_6, c_3) = \{Letizia\}$

# Definitions

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

**A context may contain other contexts (*nested context*).**

*Well-defined context* ***c*****:**

- **(Unique reference) For all *o, o′* in *c, o <> o′ => there* are *r* in *refs(o,c)* and *r′* in *refs(o′,c)* such that *r <> r′***
- **(Acyclicity) *c* is not recursively contained in *c*, nor is any of its subcontexts**
- **Information base (*IB*): a special context that** *recursively contains all the others*
- *Axiom:* *the IB is well-defined*

# Definitions
(Theodorakis, Analyti, Constantopoulos, Spyratos)

- *Synonyms:* **two different references of the same object w.r.t. the same or different contexts (external identification)**
- *Homonyms:* **two different objects which have a common reference w.r.t the same or different contexts**
  - **If these two objects are contained in a well defined context c, then there must be a unique reference w.r.t c**
  - **Since IB is assumed to be well-defined, such a context c always exists**
- *Anonyms:* **an object is anonymous w.r.t. a context c if it does not have any (direct) reference w.r.t. c. This is OK insofar as** *there is* **some context where** ***o*** **is recursively named, i.e.** ***`refs(o,c)`*** **in nonempty**

# Operations

(Theodorakis, Analyti, Constantopoulos, Spyratos)

- *Context creation:* ***CreateCXt(l)*** **takes a lexicon** ***l*** **as input and creates a context** ***c*** **s.t.** ***lex(c)=l*****, e.g.** $CreateCXt(o_1{:}Letizia,\ c_1{:}department)$ ***creates*** $c_{10}$

**with** $lex(c_{10}) = \{o_1{:}Letizia,\ c_1{:}department\}$

- *Current context setting:* ***SCC(r)*** **takes a reference** ***r*** **to a context** ***c*** **as input and sets the current context to** ***c*****, e.g.,** ***SCC(@)=IB,*** **and** $SCC(@.DEI)=c_1$

# Operations

(Theodorakis, Analyti, Constantopoulos, Spyratos)

- *Lookup:* ***`lookup(r)`*** takes a reference ***`r`*** as input and returns the set of objects ***`o`*** such that ***`r`*** belongs to ***`refs(o,c)`*** (i.e., ***`r`*** is a name of ***`o`*** in ***`c`***). ***`c`*** is either the ***`IB`*** (if ***`r`*** is absolute) or the current context
- *Insert an object into a context :* ***`Insert(o,N,r)`*** where ***`N`*** is a set of names, ***`r`*** a reference to context ***`c`*** :
  - inserts ***`o:N`*** into the lexicon of ***`c`***, if ***`o`*** is not contained in ***`c`***,
  - Adds the names contained in ***`N`*** to the ***`c`***-names of ***`o`***, otherwise
- *Delete an object from a context :* ***`DeleteObj(o,r)`*** where ***`r`*** is a reference to context ***`c`***, deletes ***`o:N`*** from the lexicon of ***`c`***

# Operations
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

- *Delete an object name from a context* : ***DeleteName(o,n,r)*** where ***r*** is a reference to context ***c***, deletes the name ***n*** from the ***c***-names of ***o***

- *Context copy:* ***CopyCXt(r)*** where ***r*** is a reference to context ***c***, returns a new context ***c'*** such that ***lex(c')= lex(c)***. e.g. ***CopyCXt(DEI)*** returns a new context $c_{11}$ s.t.:

$$lex(c_{11}) = \{\ o_1{:}prof.Tanca,\ o_5{:}professor,\ c_2{:}Databases,\ c_3{:}InformationSystems\}$$

  - Note that ***CopyCXt(r) = CreateCXt(lex(c))***

- *Context deep copy:* ***deepCopyCXt(r)*** where ***r*** is a reference to context ***c***, returns a new context ***c'*** that contains the simple objects of ***c*** and also deep copies of the contexts contained in ***c***

# Set Operations: Union
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

$r_1$ ***UNION*** $r_1$ returns a lexicon $l$ such that:

- If $r_1$ and $r_2$ are both lexicons, $l$ contains the union of the objects of $r_1$ and $r_2$, with all the object names, including all names of the common objects
- If $r_2$ is a reference to a context $c_2$, it takes the union of the two lexicons, plus the new name $r_2$ for $c_2$
- If $r_1$ and $r_2$ are references to contexts $c_1$ and $c_2$, it takes the union of both lexicons, plus the new names $r_1$ for $c_1$ and $r_2$ for $c_2$

# Example: Union

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

Let $c_1$ be the CC, then ***lex(databases) UNION lex(InformationSystems)*** returns :

- $l_1$= {$o_1$:Letizia, Tanca,
  $o_2$:head,
  $o_3$:Cristiana,
  $o_4$:Fabio, FabioAlberto,
  o6:Letizia}

while ***Databases UNION InformationSystems*** returns :

- $l_2$= {$o_1$:Letizia, Tanca,
  $o_2$:head,
  $o_3$:Cristiana,
  $o_4$:Fabio, FabioAlberto,
  $o_6$:Letizia,
  $c_2$:Databases,
  $c_3$:InformationSystems}

- Note that the last union contains, besides the two lexicons, two *views* over their objects, that is, the way they are seen from the perspectives of the two contexts $c_1$ and $c_2$

# Set Operations: Intersection

(Theodorakis, Analyti, Constantopoulos, Spyratos)

$r_1$ ***INTERSECT*** $r_1$ **returns a lexicon** $l$ **such that (let** $I$ **be the set of common objects of** $r_1$ **and** $r_2$**) :**

- **If** $r_1$ **and** $r_2$ **are both lexicons,** $l$ **contains** $I$ **plus** ***all*** **names of the common objects, plus, for all objects** $o$ **not in** $I$**, a deep copy of it deprived of all its simple objects not already in** $I$ **, plus all the relative names**
- **If** $r_2$ **is a reference to a context** $c_2$**, it takes the intersection of the two lexicons, plus** $\{c_2' : r_2\}$ **such that** $c_2'$ **is a deep copy of** $c_2$ **such that every simple object which is not in** $I$ **has been eliminated from** $c_2'$ **and from its subcontexts**
- **If** $r_1$ **and** $r_2$ **are references to contexts** $c_1$ **and** $c_2$**, it takes the intersection of the two lexicons, plus** $\{c_1' : r_1\}$ **and** $\{c_2' : r_2\}$ **as above**

# Example: Intersection

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

Let $c_1$ be the CC. ***lex(databases) INTERSECT lex(InformationSystems)*** returns :

- $l_3$= {$o_1$:Letizia, Tanca,
  $o_2$:head}

while ***Databases INTERSECT InformationSystems*** returns :

- $l_4$= {$o_1$:Letizia, Tanca,
  $o_2$:head,
  $c''_2$:Databases,
  $c''_3$:InformationSystems}

  lex($c''_2$) = {$o_1$:Letizia, $o_2$:head}

  lex($c''_3$) = {$o_2$:head, $o_1$:Tanca}

- **Note that the intersection contains both names of $o_1$, and $I=\{o_1,o_2\}$, and the two contexts $c''_1$ and $c''_2$ are copies of $c_1$ and $c_2$ where all simple objects not in $I$ have been removed**

# Set Operations: Difference
(Theodorakis, Analyti, Constantopoulos, Spyratos)

$r_1$ *MINUS* $r_2$ returns a lexicon $l$ such that (let $I$ be the set of common objects of $r_1$ and $r_2$ and $D$ be $objs(r_1)- objs(r_2)$) :

- If $r_1$ and $r_2$ are both lexicons, $l$ contains $D$ giving to D's objects the names they had in $r_1$ , plus, for all objects $o$ in $I$, which recursively contain an object of $D$, a deep copy of it deprived of all its simple objects not already in $D$ , plus all the relative names
- If $r_1$ is a lexicon and $r_2$ is a reference to a context $c_2$, then $l = r_1$ *MINUS* $lex(c_2)$
- If $r_2$ is a lexicon and $r_1$ is a reference to a context $c_1$, then $l = lex(c_1)$ *MINUS* $r_2$
- If $r_1$ and $r_2$ are references to contexts $c_1$ and $c_2$, then $l = lex(c_1)$ *MINUS* $lex(c_2)$

# Example: Difference

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

**Let** $c_1$ **be the CC.**

**RECALL:**

- $lex(c_2)= \{ o_1{:}Letizia,$
  $o_2{:}head,$
  $o_3{:}Cristiana,$
  $o_4{:}Fabio, FabioAlberto\}$
- $lex(c_3)= \{o_6{:}Letizia,$
  $o_2{:}head,$
  $o_1{:}Tanca\}$

***Databases MINUS InformationSystems*** **returns :**

$l_5= \{o_3{:}Cristiana,$
$o_4{:}Fabio, FabioAlberto\}$

# Operator properties
(Theodorakis, Analyti, Constantopoulos, Spyratos)

**Union and intersection are:**

- **Commutative**
- **Associative**
- **Distributive one w.r.t. the other**

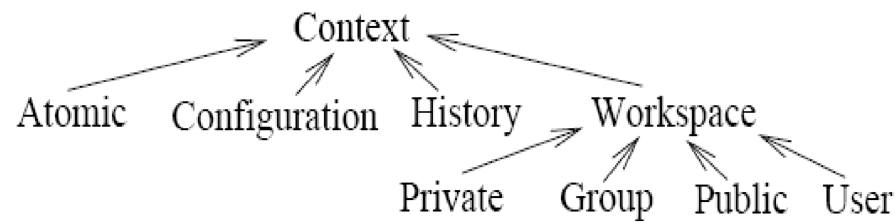
**Moreover, the following holds:**

- **(***Closure of well-definedness***) every context which is produced by means of the three operators on well-defined contexts, is also well-defined**

# Cooperation Scenario

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

Three authors (Manos, Anastasia, Nikos) are cooperating to write an article




$$lex(IB) = \begin{cases} PUBLIC : Public \\ PRIVATE : Private \\ GROUP : Group \\ HISTORY : History \\ CONFIG : Config \\ ATOMIC : Atomic \end{cases}$$

$$lex(PRIVATE) = \begin{cases} 1 : Manos \\ 2 : Anastasia \\ 3 : Nikos \end{cases}$$

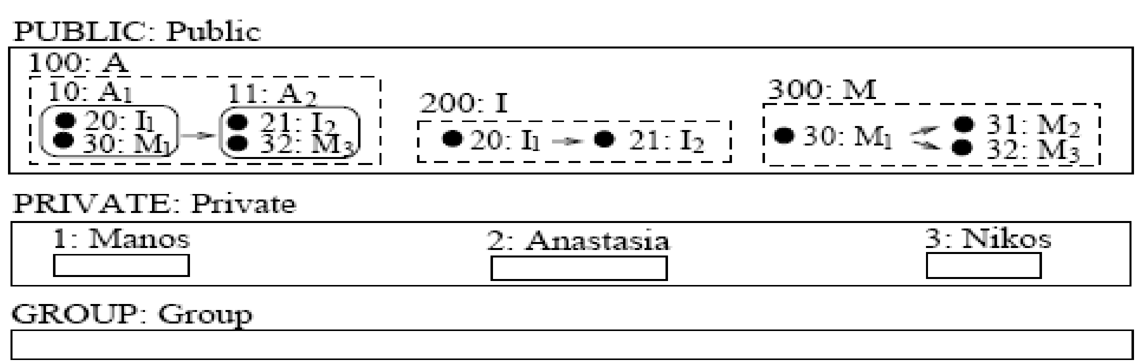
$$lex(PUBLIC) = \begin{cases} 100 : A \\ 200 : I \\ 300 : M \end{cases}$$

$$lex(HISTORY) = \begin{cases} 100 : A \\ 200 : I \\ 300 : M \end{cases}$$

$$lex(GROUP) = \emptyset$$

$$lex(CONFIG) = \begin{cases} 10 : A_1 \\ 11 : A_2 \end{cases}$$

$$lex(ATOMIC) = \begin{cases} 20 : I_1 \\ 21 : I_2 \\ 30 : M_1 \\ 31 : M_2 \\ 32 : M_3 \end{cases}$$



# Cooperation commands

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

**Derivable from the basic operations of the model**

- ***check-out(r,n):*** takes as input a reference ***r*** w.r.t. the public workspace, and a name ***n***:
  1. Copies the history context of the version referred to by r, from the public workspace into the home workspace of the user, under the same name.
  2. Copies the version referred to by r (call this version v), from the public workspace into the CC (call this copy v').
  3. Adds v' into the copy of the history context, under the name n.
  4. Updates the copy of the history context by adding a link from v to v'.

# Cooperation commands

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

**Derivable from the basic operations of the model**

- ***check-in(r,h,n):*** takes as input a reference ***r*** w.r.t. the CC, a reference ***h*** w.r.t. the public workspace, and a name ***n***, and
  - it copies the version referred to by ***r*** from the CC into the history context of the public workspace referred to by ***h***, under the name ***n***.

# Cooperation commands
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

**Derivable from the basic operations of the model**

- ***export($r_1$,$r_2$,n):*** takes as input two references $r_1$ and $r_2$, w.r.t. the CC, and a name $n$, and:

1. Creates a context (call it $c$), whose lexicon is the union of the lexicon of the context referenced by $r_1$, and the context referenced by $r_2$ (call the last context $c_2$).
2. Creates a link from the last edited version (that is, the one named Current) to the context $c_2$.
3. Context $c_2$ is assigned two names w.r.t. $c$: (a) The value of Username, to indicate the author of the version, and (b) Current, to indicate that $c_2$ is the last edited version (the name Current is then deleted from the names of the previously edited version).
4. Copies the context $c$ into the group workspace, under the name $n$.

# Cooperation commands

(Theodorakis, Analyti, Constantopoulos, Spyratos)

**Derivable from the basic operations of the model**

- ***`import(r,n):`*** takes as input a reference ***r***, w.r.t. the group workspace, and a name ***n***. Then, it :
  1. Copies the context referenced by ***r*** from the group workspace into the CC, under the name ***n***.
  2. Deletes the original context from the group workspace.

# Cooperation Scenario

## (Theodorakis, Analyti, Constantopoulos, Spyratos)

1. Commands by user Manos.
   $/* \ CC = 1, \ Home = @.Private.Manos,$
   $Username = Manos \ */$
   (a) $check\text{-}out(A.A_2, A_3)$.
   (b) $SCC(A_3)$.
   (c) $check\text{-}out(I.I_2, I_3)$.
   (d) $\cdots$.
   (e) $SCC(Home)$.
   (f) $a_2 = lookup(A.A_2)$.
   (g) $insert(\ createCxt($
   $\{(a_2 : Public, Current)\}), \{TMP\}, Home)$.
   (h) $export(TMP, A_3, A)$.

2. Commands by user Anastasia.
   $/* \ CC = 2, \ Home = @.Private.Anastasia,$
   $Username = Anastasia. \ */$
   (a) $import(A, TMP)$.
   (b) $check\text{-}out(A.A_2, A_3)$.
   (c) $SCC(A_3)$.
   (d) $check\text{-}out(I.I_2, I_3)$.
   (e) $check\text{-}out(M.M_3, M_4)$.
   $\cdots$
   (f) $SCC(Home)$.
   (g) $export(TMP, A_4, A)$.

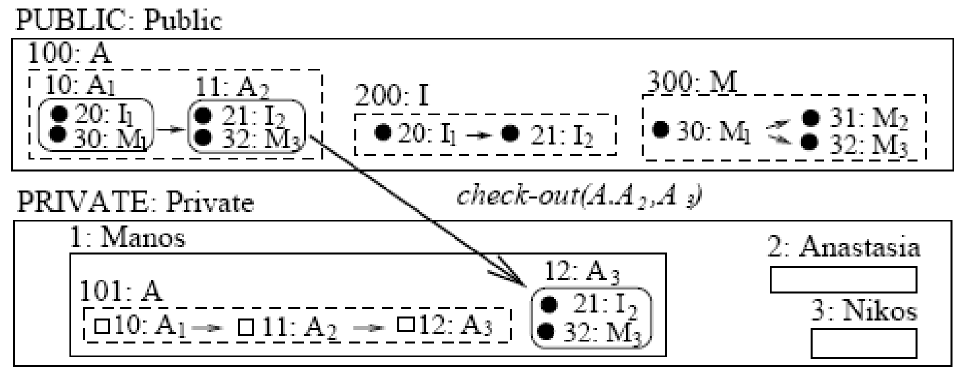
3. Commands by user Nikos.
   $/* \ CC = 3, \ Home = @.Private.Nikos,$
   $Username = Nikos. \ */$
   (a) $import(A, TMP)$.
   (b) $copy(A.Current, A_5)$.
   $\cdots$
   (c) $check\text{-}in(A_5, TMP, A_3)$.

Commands by the various users who interact with the workspaces

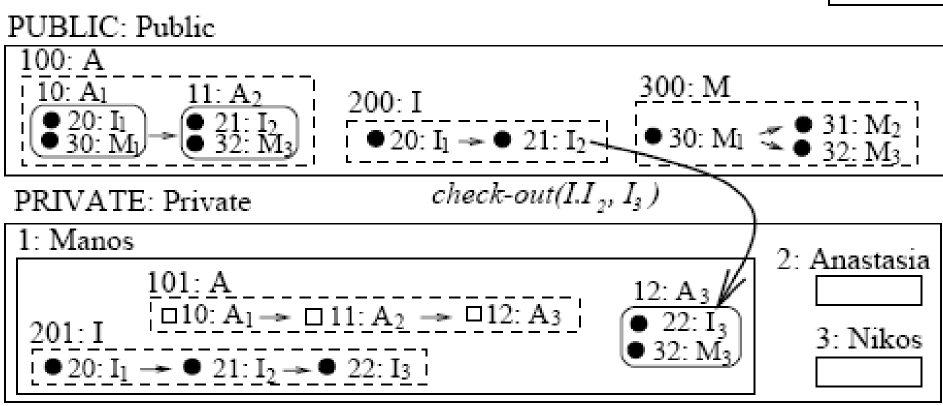
# Cooperation Scenario
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

Manos' interactions with the various workspaces

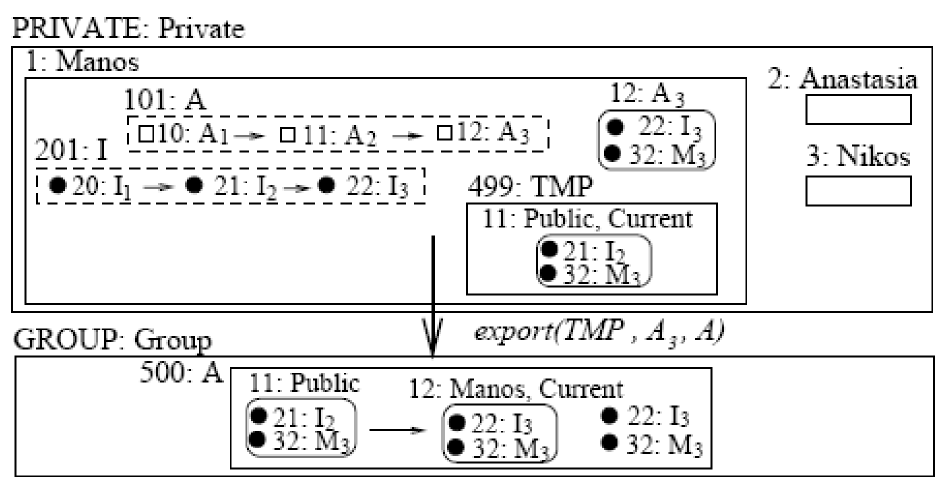
(a)




(b)



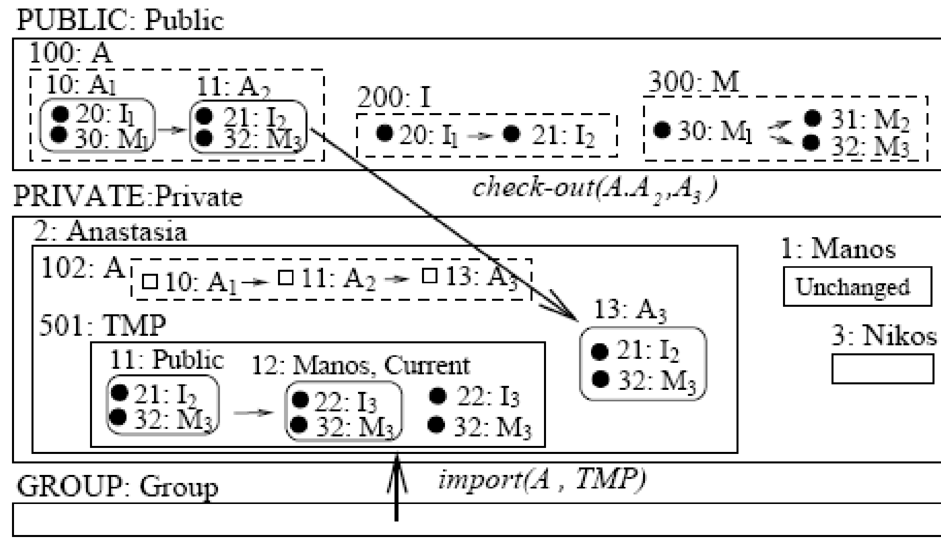

(c)



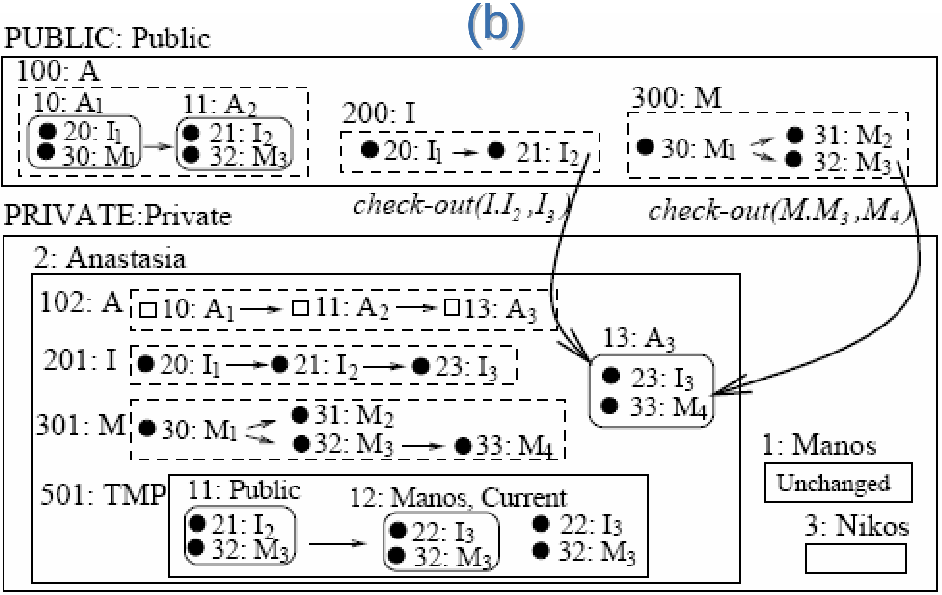
# Cooperation Scenario
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

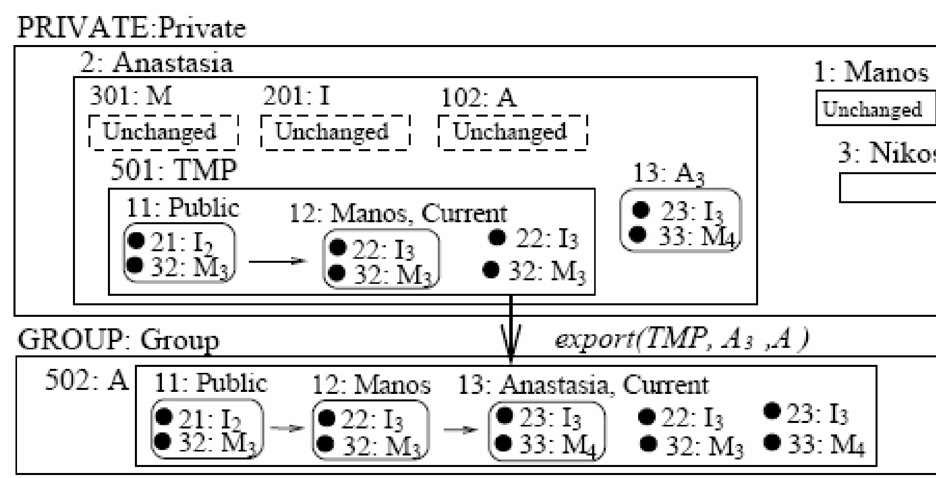
Anastasia's interactions with the various workspaces

(a)




(b)

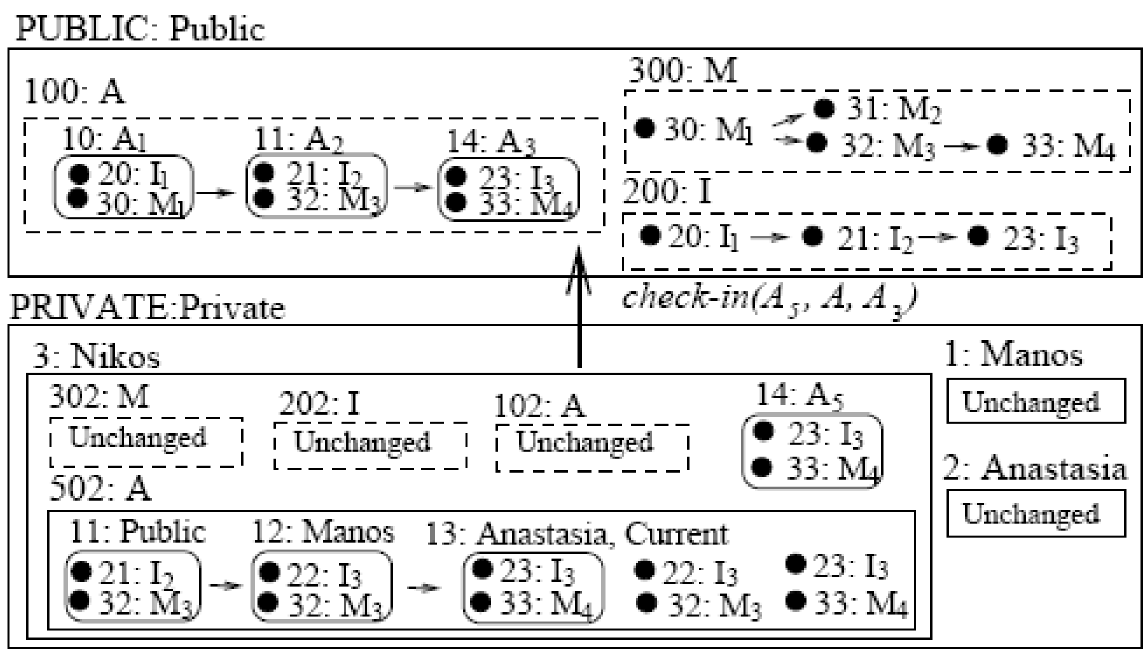



(c)

# Cooperation Scenario
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

Nikos' check-in into the public workspace



# Context in Information bases
## (Theodorakis, Analyti, Constantopoulos, Spyratos)

- The commands check-in, check-out, import and export, are examples of simple communication commands that can be implemented using the basic operations of the model.
- In a more complex environment, like in a software engineering project, where several groups are developing software in parallel, a coordinating unit may need to compare modules coming from various groups, before merging them into a single module.
- Such information can be obtained through more sophisticated higher level commands that can also be implemented using the basic operations of the model.
- The Information Base can be organized in a number of different ways.
- Choosing the appropriate organization is THE design problem that depends on the application.
- A methodology for information base development is needed

# Context Relational Model

(Roussos, Stavrakas,Pavlaki)

## Context Relational model:

- the information provider needs to specify the context under which information becomes relevant
- information users specify their own current context when requesting data, in order to denote the part that is relevant to their specific situation
- management of context should take place at the level of database systems in a uniform way and consequently context should be treated as a first-class citizen in data models and query languages
- Examples:
  - **a product (e.g. car, dvd) whose specification changes according to the country it is being exported to**
  - **a Web page that is to be displayed on devices with different capabilities**
  - **a report that must be represented at various degrees of detail and in various languages**

# Context Relational Model

(Roussos, Stavrakas,Pavlaki)

- **context introduced to deal with the ambiguity of data interpretation under different environmental conditions, such as current position of the user or the media he is using (laptop, mobile, PDA).**
- **extends the relational model to deal with context**
- **context is treated as first-class citizen at the level of database models and query languages.**
- **an attribute may** *not exist under some contexts* **or** *have different values under different contexts*
- **they also have a set of basic operations which extend relational algebra so as to take context into account**

# Context Relational Model

(Roussos, Stavrakas,Pavlaki)

- *Information entities* manifest different *facets*, whose contents can vary in structure and value
- Each facet is associated with a context, stating the conditions under which this facet holds
- *dimensions*: the set of parameters used to specify the world
- *context specifier:* a syntactic construct used to qualify pieces of data and specify sets of worlds (or contexts) under which these pieces hold
- it is possible to have at the same time variants (facets) of the same information entity, each holding under a different set of worlds (context)

# Context Relational Model

(Roussos, Stavrakas,Pavlaki)

***D*** nonempty set of dimension names, for each ***d*** in ***D***, let $V_d$ be the domain of ***d***, with $V_d$ non empty:

- A world ***w*** with respect to ***D*** is a set of pairs ***(d,v)***, such that for every ***d*** in ***D*** exactly one ***(d,v)*** belongs to ***w***. E.g. the following are context specifiers:

1. `[device=PC]`
2. `[device=PDA, payment in {credit card, cash}]`

- `{(device; PC)}` is the world of context 1, while `{(device, PDA); (payment, credit card)}` and `{(device, PDA); (payment, cash)}` are the worlds of context 2.
- It is not necessary for a context specifier to contain values for every dimension in D. Omitting a dimension means that its value may range over the whole dimension domain

# Context Relational Model

(Roussos, Stavrakas, Pavlaki)

- Let **$c_1$**, **$c_2$** be two context specifiers:
  - ***$c_1$* UNION *$c_2$*** is the context specifier containing the worlds belonging either to ***$c_1$*** or to ***$c_2$***
  - ***c1 INTERSECT c2*** is the context specifier

    ***c3 ={w1 ∩ w2 | w1 belongs to c1 and w2 belongs to c2}***
- context specifer **[]** is a *universal context* and represents the set of all possible worlds
- context specifier **[-]** is an *empty context* and represents the empty set of worlds

# Example: Web site about digital cameras (Roussos, Stavrakas,Pavlaki)

- For each camera: brand name, model, a picture, size in megapixels and price.
- Customers connect to this Web site using a variety of devices ranging over desktop computer (PC), PDA and cell phone.
- Customers can select the method of payment between Credit Card and Cash.
- A customer using a PDA receives a picture of lower resolution than when using a desktop computer. When using a cell phone no picture exists and only textual information is provided.
- The price of a digital camera varies according to the payment method.

# Example: Web site about digital cameras (Roussos, Stavrakas,Pavlaki)

- DCAMERA( Brand,Model, MPix, Photo, Price)
- Dimensions:
  - device, ranging over {PC, PDA, CELL};
  - payment, ranging over {credit card, cash}

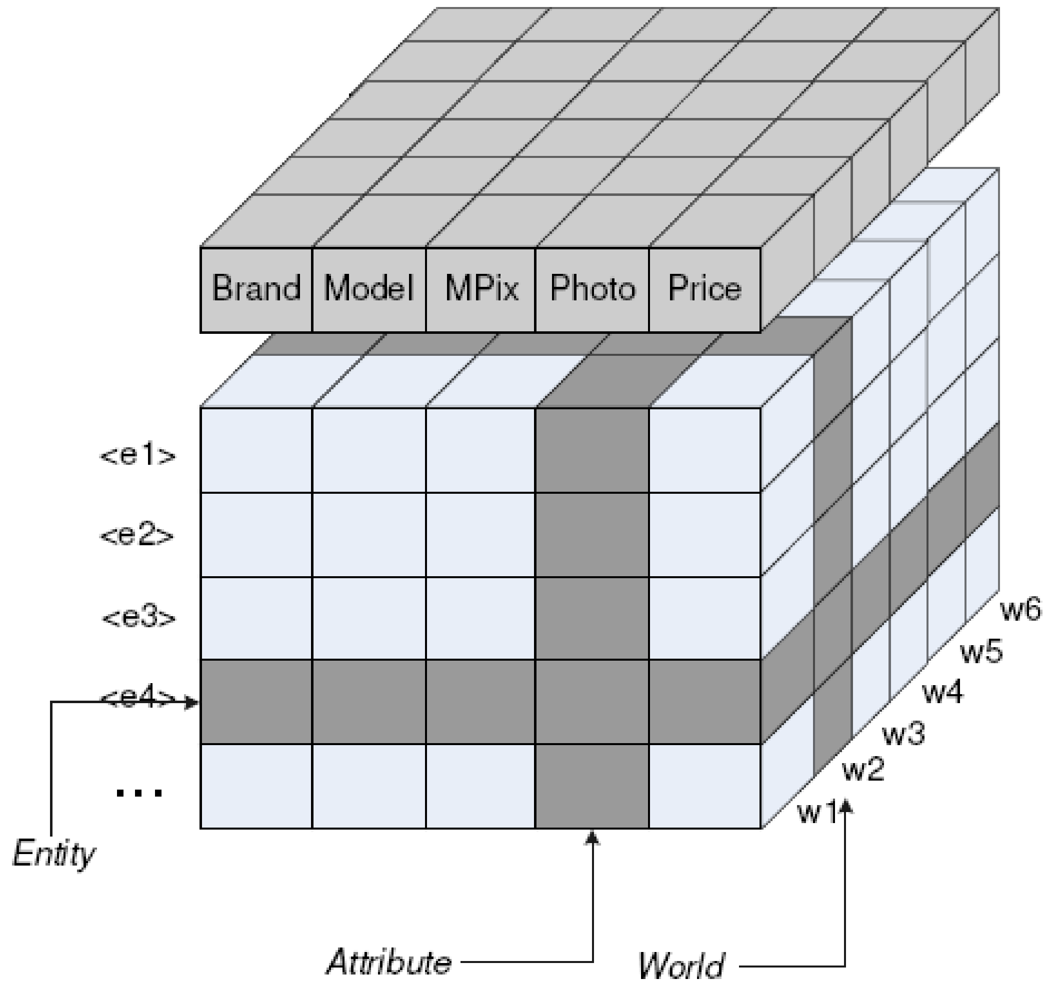
| World | Device | Payment |
|---|---|---|
| $w_1$ | PC | Credit Card |
| $w_2$ | PDA | Credit Card |
| $w_3$ | CELL | Credit Card |
| $w_4$ | PC | Cash |
| $w_5$ | PDA | Cash |
| $w_6$ | CELL | Cash |

| Attributes | Worlds |
|---|---|
| Brand | defined in every world |
| Model | defined in every world |
| MPix | defined in every world |
| Photo | defined only for worlds with browsing device in {PC,PDA} |
| Price | defined in every world but its values may change |

# Context Relational Model

## (Roussos, Stavrakas, Pavlaki)

- *Information entities* are *multi-facet*
- Each facet $\mathbf{f_{i,j}}$ is the variant of an entity
- A set of entities is a *context relation*
- For a context relation, a number of (possibly different) attributes is defined for each possible world
- the value of an attribute $\mathbf{A_i}$ in world $\mathbf{w_j}$ is denoted by $\mathbf{A_i\{w_j\}}$



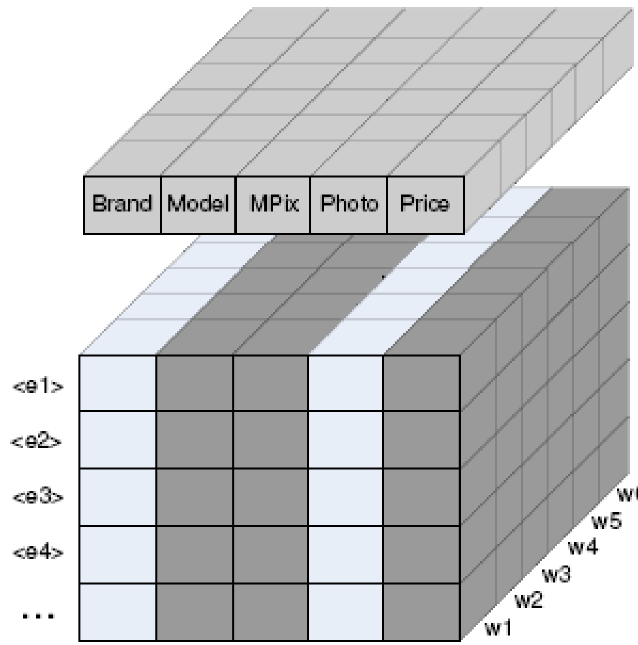
# CR Model Operations

## (Roussos, Stavrakas, Pavlaki)

<u>*context-project :*</u>

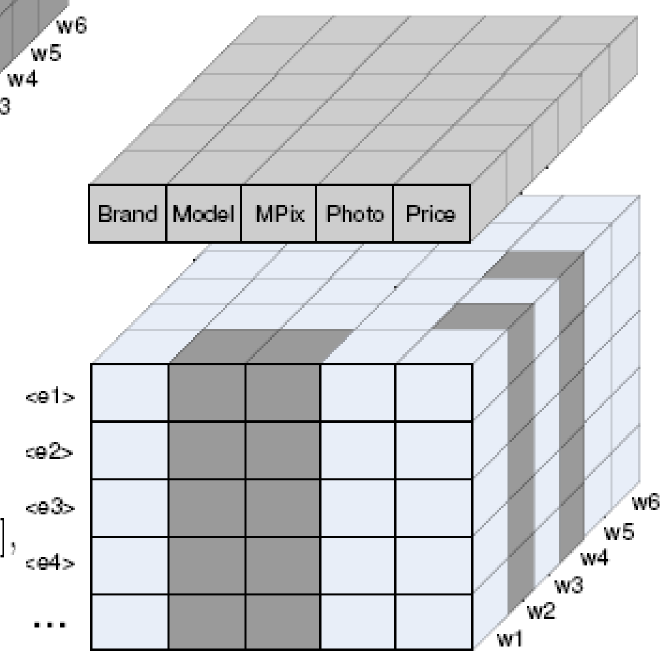
- the output contains only the attributes specified in the condition.
- The result is a new *context-relation* that has only the vertical slices that correspond to the projected attributes.
- The resulting context relation will still include slices for all possible worlds, however will only contain actual values for the projected part.

$$\pi_{Model[\,],\,MPix[\,],\,Price[\,]}\,dcamera$$




**Context-Project for attributes Model, MPix and Price in all worlds**

**Context-Project for attributes Model and MPix in world w1 and for Price in worlds {w2,w4}**



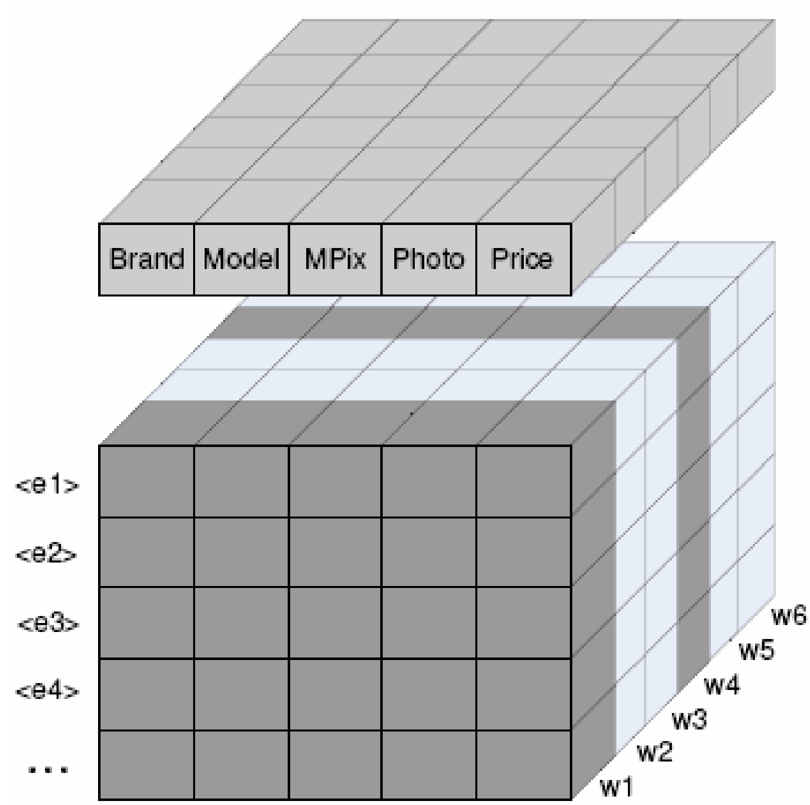

$$\pi_{Model[Device=PC,Payment=CreditCard],MPix[Device=PC,Payment=CreditCard],\,Price[Device=PDA,Payment\ in\ \{CreditCard,Cash\}]}\,dcamera$$

# CR Model Operations
(Roussos, Stavrakas,Pavlaki)

*World Project:*

retains only those facets of entities that hold under specified worlds

for a customer using a PC the relevant worlds are w1 and w4




$\kappa_{[w_1, w_4]} dcamera$

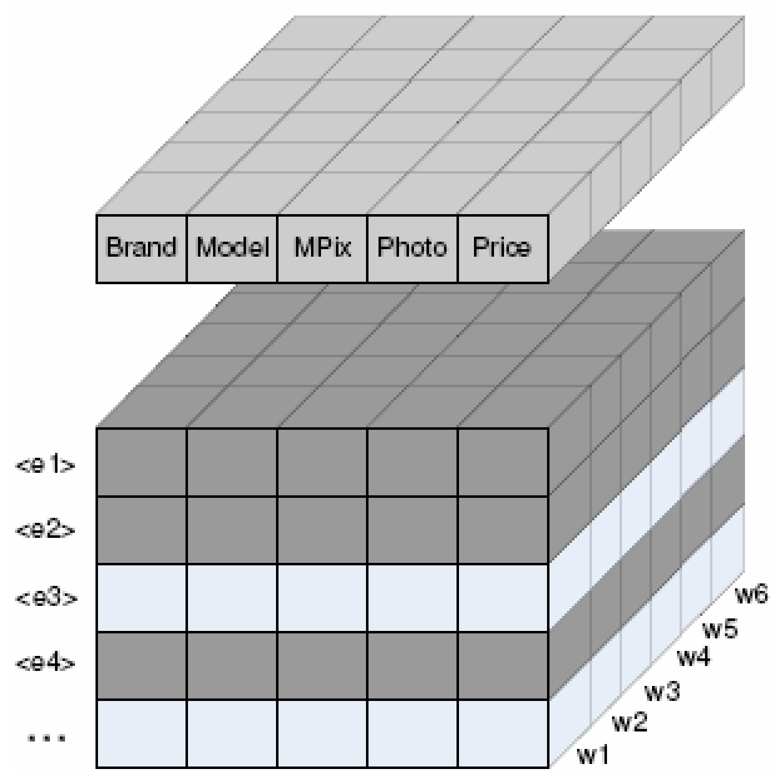
# CR Model Operations
## (Roussos, Stavrakas, Pavlaki)

*Entity context-select:*

uses context in order to express conditions that involve attribute values under different worlds

$$\sigma^{entity}_{(BRAND[\,]='Kodak'\ AND\ MPix[\,]>3)}\, dcamera$$

digital cameras created by Kodak, with more than three megapixels.



# CR Model Operations

(Roussos, Stavrakas, Pavlaki)

## *Entity context-select examples:*

$$\sigma^{entity}_{(BRAND[\,]='Kodak'\ AND\ Price[Device=PC,Payment=Cash]\ <\ 250)}{}^{dcamera}$$

only selects entities whose price is less than 250 when the customer is paying in cash

cross-world query, compares the values of the same attribute in different worlds, asking for cameras that have lower price if the customer pays in cash rather than using a credit card

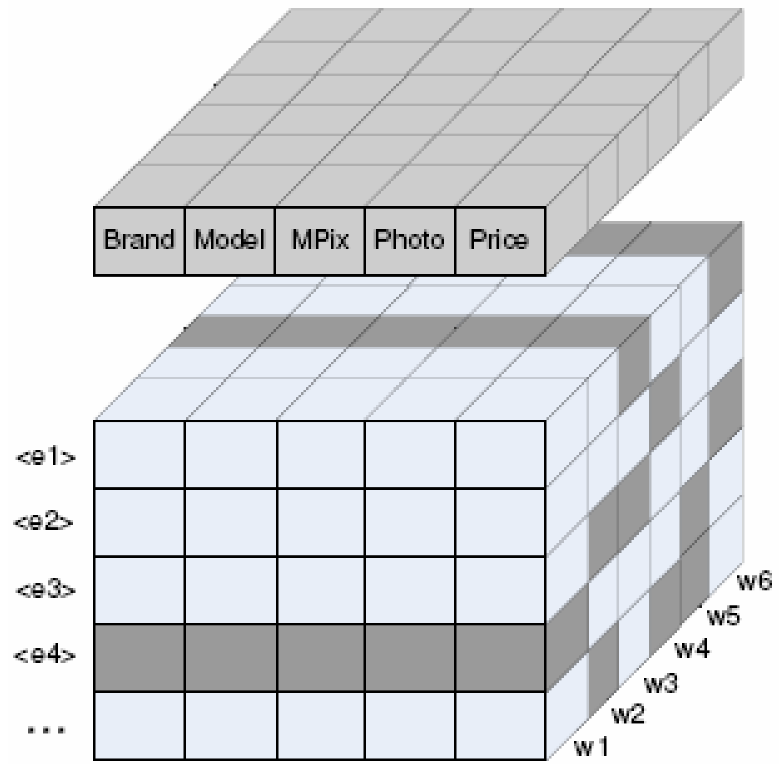
$$\sigma^{entity}_{(Price[Device=PC,Payment=Cash]\ <\ Price[Device=PC,Payment=CreditCard])}{}^{dcamera}$$

# CR Model Operations
(Roussos, Stavrakas,Pavlaki)

*Facet context-select:*

selects only the facets of an entity that satisfy the condition, instead of the whole entity as the $\sigma^{entity}$ operator.

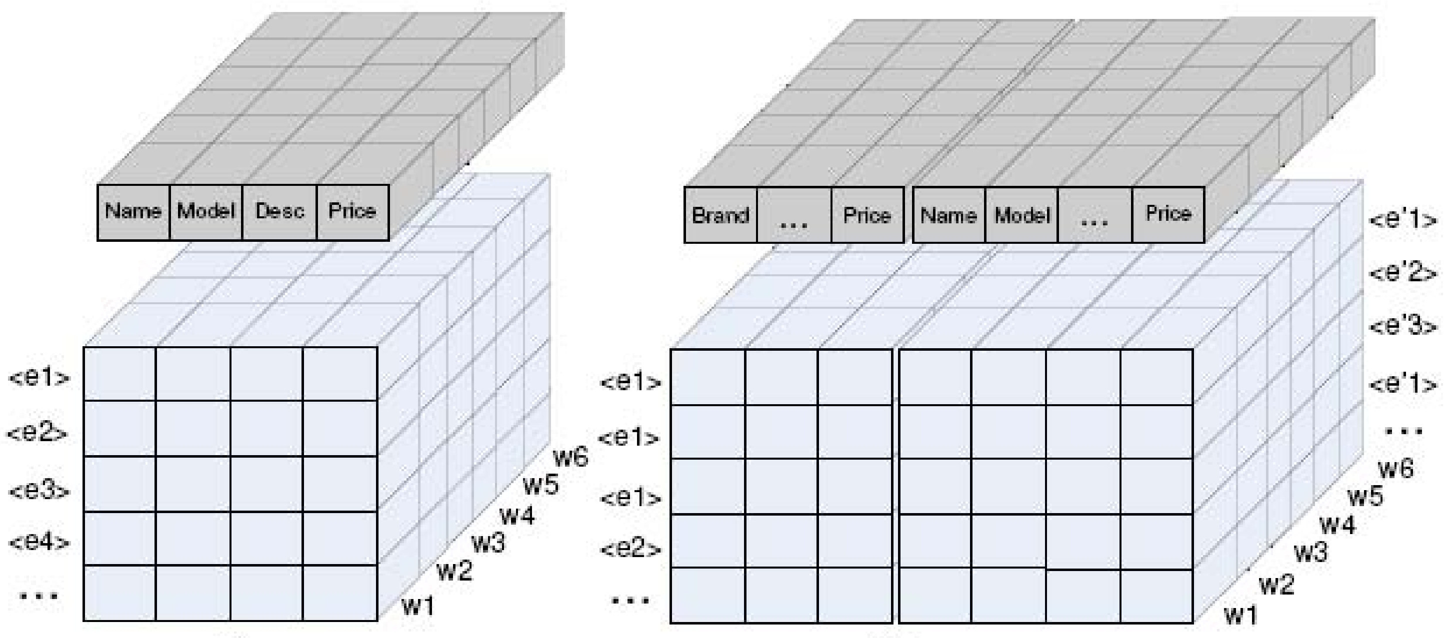
the result of selecting facets with Price less than 500 euros



# CR Model Operations
## (Roussos, Stavrakas,Pavlaki)

<u>*context cartesian product:*</u>

- creates new *context-relations* from existing ones
- (a) is a new *context-relation*
  **`accessories(name,model,description,price)`**
- (b) is the cartesian product




(a) (b)

# Derived Operations
(Roussos, Stavrakas,Pavlaki)

*context join:*

- defined as the composition of the *context cartesian product* and the *entity context-select* operations
- E.g., Kodak digital cameras with cheap 200mm lens (e.g. where **`accessories: Price < 0.2 * dcamera:Price`**).

*context natural join:*

- It is necessary for selecting entities where **`dcamera.Model = accessories.Model`** for all worlds

*union, intersection, difference and division*

- are defined only for *context-relations* that have exactly the same attributes defined under each world. The semantics are the same as in set theory, but with entities as the basic elements

# Example

(Roussos, Stavrakas,Pavlaki)

- Customer uses a PC and wants to buy a Kodak camera costing less than 400 Euros
- He/she also wants to buy accessories for this camera, so for all selected cameras he/she then requests to see all available accessories.
- He/she asks the price using cash for camera and credit card for accessories.
- In database terms he/she requests **`dcamera.Price`** for cash and **`accessories.Price`** for credit card:

$$\texttt{Ctx-Rel1} \leftarrow \textit{context cartesian product}(dcamera, accessories)$$

$$\texttt{Ctx-Rel2} \leftarrow \sigma^{entity}_{(dc.Model[\,] = ac.Model[\,]\ AND\ BRAND[\,]='Kodak'\ AND\ dc.Price[Device=PC,\ Payment=Cash]<400)}\texttt{Ctx-Rel1}$$

$$\texttt{Result} \leftarrow \pi_{dcamera.Model[Device=PC,Payment=Cash],\ dcamera.Price[Device=PC,Payment=Cash],\ accessories.Name[Device=PC,Payment=CreditCard],\ accessories.Price[Device=PC,\ Payment=CreditCard]}\texttt{Ctx-Rel2}$$

# CR Model: conclusions

(Roussos, Stavrakas,Pavlaki)

- In the *CR* model, a world slice $w_i$ is a classical relation as in the relational model.
- The tuples of this relation would correspond to the facets of the entities for this world, BUT:
- if we decompose a *context-relation* to a series of relations, the link between facets that consist a single information entity is lost.
- This link is used in the *CR* model to formulate cross-world queries.

# Adding Context to Preferences

(Stefanidis, Pitoura and Vassiliadis)

- **a** *preference database* **system that supports context-aware queries, that is,** *queries whose results depend on the context at the time of their submission*
- **context is modeled as** *a set of multidimensional attributes.*
- **data cubes used to store the dependencies between context-dependent preferences and database relations and OLAP techniques for processing context-aware queries**
- **Manipulation of the captured context data at various levels of abstraction**
  - **E.g., in the case of a context parameter representing location, preferences may be expressed at the levels: city, region, country etc.**
- **auxiliary data structure, called context tree, stores results of past context-aware queries** *indexed by the context of their execution*

# Adding Context to Preferences

(Stefanidis, Pitoura and Vassiliadis)

- **users express their preferences on specific attributes of a relation**
- **preferences may have different values** *depending on context*
- **a** *context state* **corresponds to an assignment of values to** *context parameters*
- **different levels of abstraction for the context data introduced by allowing context parameters** *to take values from hierarchical domains*
- *basic preferences*, **i.e., preferences associating database relations with a single context parameter, are combined to compute** *aggregate preferences* **that include more than one context parameter**

# Adding Context to Preferences

## (Stefanidis, Pitoura and Vassiliadis)

- **context descriptors used to express preferences on specific database instances for a variety of context states expressed with varying levels of detail**
- *context resolution problem:* **identifying those preferences whose context states are the most relevant to the context state of the query. Divided into two steps:**
  - Identification of all the candidate context states that encompass the query state
  - selection of the most appropriate state among these candidates

# Adding Context to Preferences

(Stefanidis, Pitoura and Vassiliadis)

- **Running example:**

*Restaurant ( rid, name, phone, region, cuisine)*

*User ( uid, name, phone, address, e-mail)*

- two relevant context parameters: **`location`** and **`weather`**.
- preferences about restaurants expressed by providing a numerical score between 0 and 1 that quantifies the degree of interest for a restaurant
- the degree of interest of a user for a restaurant depends on the values of the two relevant context parameters

e.g. user Mary may give to restaurant *Zoloushka* that serves "Russian" food a higher score when the weather is rainy than when the weather is sunny

# Adding Context to Preferences

## (Stefanidis, Pitoura and Vassiliadis)

*Restaurant(rid, name, phone, region, cuisine)*

*User(uid, name, phone, address, e-mail)*

- also the current user's location affects the result of a query, e.g., a user may prefer restaurants that are nearby her current location.
- The user provides preference scores that depend on **location** and preference scores that depend on **weather**
- These basic preferences are combined to produce an aggregate score that depends on more than one context parameter

# Adding Context to Preferences
(Stefanidis, Pitoura and Vassiliadis)

## *Modeling Context*

- assume a countable collection of *attribute names*. Each attribute $\mathbf{A_i}$ is characterized by a name and a domain $\mathbf{dom(A_i)}$
- Context is modeled through a finite set of special-purpose attributes, called *context parameters*
- For a given application **X**, its *context environment* **CX** is a set of **n** context parameters $\{c_1, c_2, \ldots, c_n\}$
- a *context state* is an assignment of values to context parameters. The context state at time instant **t** is a tuple with the values of the context parameters at time instant **t**, $CSX(t) = \{c_1(t), c_2(t), \ldots c_n(t)\}$, where $c_i(t)$ is the value of the context parameter $c_i$ at **t**.

e.g., assuming location and weather as context parameters, a context state is:

**CS(current) = {Acropolis, sunny}**

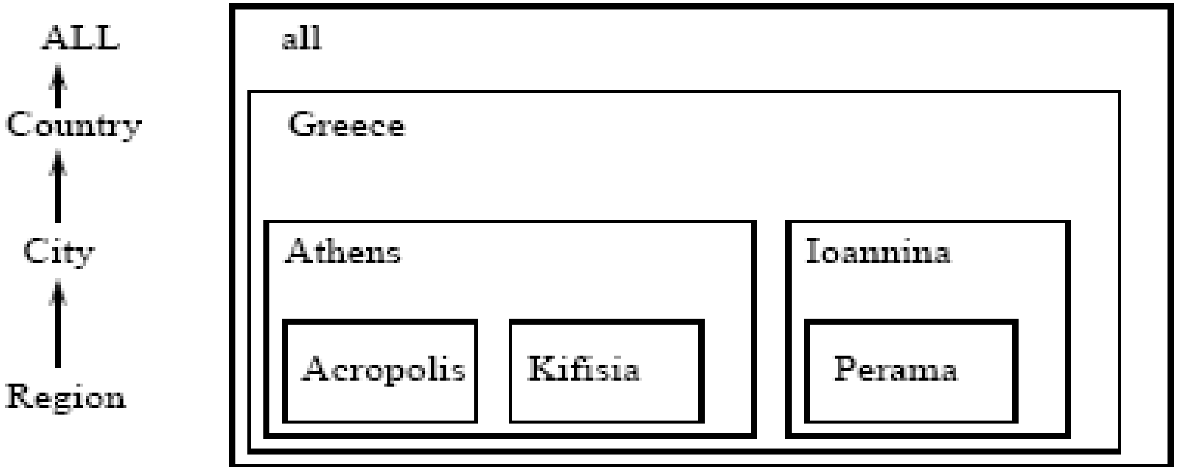
# Adding Context to Preferences

## (Stefanidis, Pitoura and Vassiliadis)

- two kinds of context parameters:
  - *static context parameters* take as value a simple value out of their domain
  - *dynamic context parameters* are instantiated by the application of a function, the result of which is an instance of the domain of the context parameter

E.g.: **`weather`** is a static parameter, i.e., each new value for weather is derived by an explicit update. **`Location`** is a dynamic parameter defined as a function of time. Its value can be computed at the needed moment without the need for continuous explicit updates

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)




Hierarchies of context attributes: the `location` hierarchy

# Adding Context to Preferences

(Stefanidis, Pitoura and Vassiliadis)

## *Modeling Preferences*

- *Basic Preferences:* described by
  - a context parameter $\mathbf{c_i}$,
  - a set of non-context parameters $\mathbf{A_i}$,
  - a degree of interest, i.e., a real number between **0** and **1**.
- So, for the context parameter $\mathbf{c_i}$, we have:

  $$\mathbf{preferencebasic_i\ (c_i, A_{k+1}, \ . \ . \ . \ , A_n) = interest\ score_i}.$$

- In the running example there are two context parameters, location and weather, and a set of non-context parameters: the attributes about restaurants and users. E.g.:

**preferencebasic$_1$(Acropolis,BeauBrummel,Mary) = 0.8**

**preferencebasic$_2$ (cloudy,BeauBrummel,Mary) = 0.9**

# Adding Context to Preferences
(Stefanidis, Pitoura and Vassiliadis)

## *Modeling Preferences*

- *Aggregate Preferences:* described by:
  - **a set of context parameters $c_i$**
  - **a set of non-context parameters $A_i$ and a degree of interest:**

$$\texttt{preference}(c_1, \ldots c_k, A_{k+1}, \ldots, A_n) = \texttt{interest score}$$

- The interest score of the aggregate preference is a *value function* of the individuals scores (the degrees of the basic preferences).

# Adding Context to Preferences

(Stefanidis, Pitoura and Vassiliadis)

## *Modeling Preferences*

- The value function prescribes how to combine basic preferences to produce the aggregate score, according to the user's profile (e.g. value functions can be based on a weighted average of the simple preferences)
- Users define in their profile how the basic scores contribute to the aggregate ones, e.g. by giving a weight to each context parameter. So, if the weight for a context parameter is $\mathbf{w_i}$ and $\mathbf{interest\ score_i}$ is the score defined by the associated basic preference, then the aggregate interest score will be:

$$\mathbf{interest\ score = w_1 \times interest\ score_1 + \ldots + w_k \times interest\ score_k}$$

if the weight of location is **0.6** and the weight of weather is **0.4**, the preference has score:

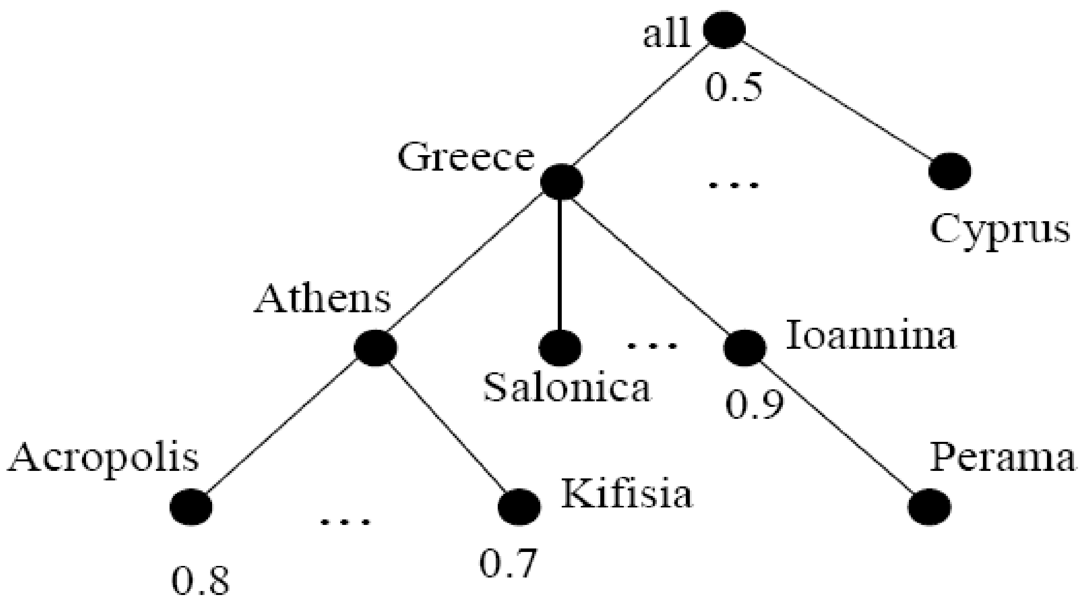
$$\mathbf{0.6 \times 0.8 + 0.4 \times 0.9 = 0.84}$$ . That is:

**preference(Acropolis, cloudy,BeauBrummel,Mary) = 0.84.**

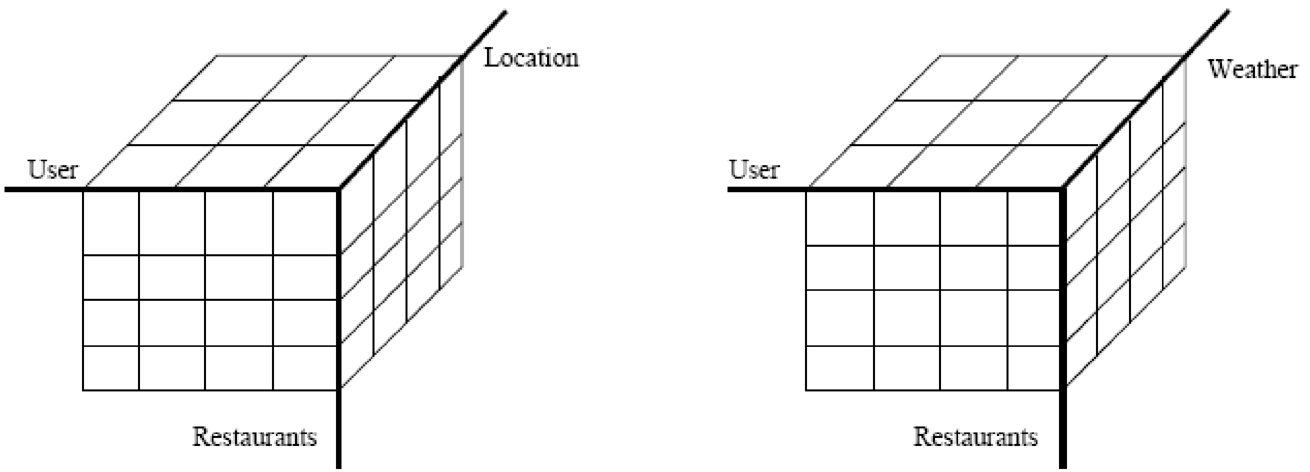
# Adding Context to Preferences
(Stefanidis, Pitoura and Vassiliadis)

- simple preferences stored in OLAP in data cubes
- this allows to aggregate data along a hierarchical context parameter, e.g., grouping preferences for all cities of a specific country
- Aggregate preferences are not explicitly stored.
- To improve performance, aggregate preferences computed as results of previous queries are stored into an auxiliary data structure called *context tree*.
- A path in the context tree corresponds to an assignment of values to context parameters, that is, to a context state, for which the aggregate score has been previously computed
- Results stored in a context tree are re-used to speed-up query processing.

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)

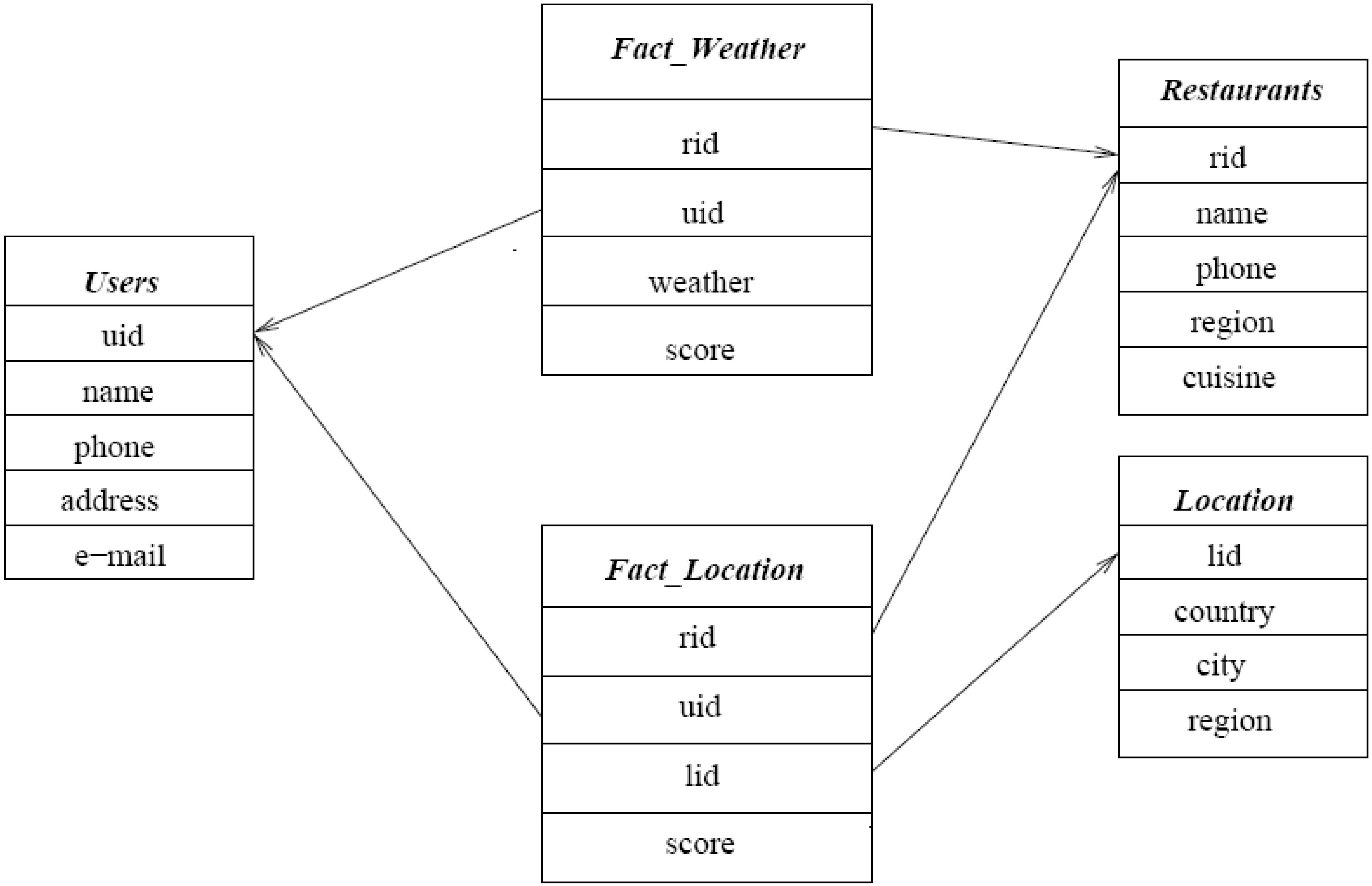



The hierarchy of locations




Data cubes for each context parameter

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)




The two fact tables and the dimension tables for Users and Restaurants

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)

| G_ID | Region | City | Country |
|---|---|---|---|
| 1 | Acropolis | Athens | Greece |
| 2 | Kefalari | Athens | Greece |
| 3 | Perama | Ioannina | Greece |
| ... | | | |

Typical dimension table

| G_ID | Region | City | Country | Level |
|---|---|---|---|---|
| 1 | Acropolis | Athens | Greece | 1 |
| 2 | Kefalari | Athens | Greece | 1 |
| 3 | Polichni | Salonica | Greece | 1 |
| ... | | | | |
| 101 | NULL | Athens | Greece | 2 |
| 102 | NULL | Salonica | Greece | 2 |
| ... | | | | |
| 120 | NULL | NULL | Greece | 3 |
| 121 | NULL | NULL | Cyprus | 3 |
| ... | | | | |

Extended dimension table, used to be able to store also preferences at higher aggregation levels

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)

*SELECT R.name, FL.score*
*FROM Users U, Restaurants R, Fact_Location FL, Location L*
*WHERE U.uid = FL.uid AND R.rid = FL.rid AND L.lid = FL.lid AND U.name ='Mary' AND L.location ='Acropolis'*
*ORDER BY FL.score DESC;*

Query with basic preferences: **Look for Mary's most preferable restaurants near Acropolis, independently of the status of weather.**

# Adding Context to Preferences
## (Stefanidis, Pitoura and Vassiliadis)

$SELECT\ R.name,\ FL.score$
$FROM\ \ Users\ \ U,\ \ Restaurants\ \ R,\ \ Fact\_Location\ \ FL,$
$Location\ L$
$WHERE\ \ U.name\ \ =' Mary'\ \ AND\ U.uid\ \ =\ \ FL.uid$
$AND\ \ R.rid\ \ =\ \ FL.rid\ \ AND\ \ L.lid\ \ =\ \ FL.lid\ \ AND$
$current\_location\ =' Acropolis';$

Subquery 1

$SELECT\ R.name,\ FW.score$
$FROM\ Users\ U,\ Restaurants\ R,\ Fact\_Weather\ FW$
$WHERE\ U.name\ =' Mary'\ AND\ U.uid = FW.uid\ AND$
$R.rid = FW.rid\ AND\ current\_weather\ =' sunny';$

Subquery 2

Query with aggregate preferences: **Look for Mary's most preferable restaurants <u>in the current context</u>.**

Aggregate scores for restaurants are computed using the value function

# The layered view model (LVM)

(Rajugan, Chang, Dillon, Feng)

**Based on two postulates about the real world:**

- *Postulate 1*: **The term** *context* **refers to** *the domain that interests an organization as a whole*. **It is more than a measure and implies a meaningful collection of objects, relationships among these objects, as well as some constraints associated with the objects and their relationships, which are relevant to its applications. For example, people, order, and Bounded customer can be examples of context in the e-Sol system.**
- *Postulate 2*: **The term** *view* **refers to** *a certain perspective of the context* **that makes sense to one or more stakeholders of the organization or an organization unit at a given point in time**

# The layered view model (LVM)

## (Rajugan, Chang, Dillon, Feng)

**Three levels of abstraction:**

- **The** *conceptual level* **describes the structure and semantics of XML views in a way which is more comprehensible to human users. It hides the details of view implementation and concentrates on describing objects, relationships among the objects, as well as the associated constraints upon the objects and relationships. Modeling primitives include** *object, attribute, relationship***, and** *constraint.* **Conceptual views are modeled by using** *UML/OCL (Object Constraint Language, OMG)*
- **The** *logical level* **describes the schema of XML views for the view implementation, using the XML Schema language. Views at the conceptual level are mapped into the views at the schema level via appropriate transformation mechanism (e.g. UML to XML Schema by the same authors)**
- **the** *document, or instance level* **implies a fragment of instantiated XML data, which conforms to the corresponding view schema defined at the upper level. Here, the conceptual operators are mapped to query expressions (e.g. XQuery) An XML instance view is an instantiated imaginary XML document which conforms to the XML schema view defined at the schema level.**

# The layered view model (LVM)

(Rajugan, Chang, Dillon, Feng)

## Conclusions

- Contexts are used to provide different views of the same object or group of objects
- A less formal approach than the previous ones
- Methodological considerations are provided
- Strongly based on UML and XML formalisms

# Conclusions

- Even within Information Systems, context is used for different purposes:
  - to provide access to different facets of the same object or group of objects
  - to provide/equip different users with specific functions in various situations
  - to tailor data or services in different "shapes"
  - to associate data with different preference for values in different situations
- Fundamental underlying concepts:
  - VIEW
  - CONTEXT DIMENSION

# bibliography


- A. Segev and A. Gal: Putting Things in Context: A Topological Approach to Mapping Contexts to Ontologies, S. Spaccapietra et al. (Eds.); Journal on Data Semantics IX, LNCS 4601, pp. 113-140, 2007.Springer-Verlag Berlin Heidelberg 2007
- K. Stefanidis, e. Pitoura and p. Vassiliadis: A Context-Aware Preference Database System, J. PERVASIVE COMPUT. & COMM. 1 (1), March 2005, Troubador pub. Ltd.
- M. Theodorakis, A. Analyti, P. Constantopoulos, N. Spyratos: A theory of contexts in information bases. Inf. Syst. (IS) 27(3):151-191 (2002)
- R. Motschnig-Pitrik: A Generic Framework for the Modeling of Contexts and its Applications, Data & Knowledge Engineering, 2000.
- Y. Roussos Y. Stavrakas V. Pavlaki: Towards a Context-Aware Relational Model, In "Contextual Representation and Reasoning" Workshop (CRR'05), held in conjunction with CONTEXT'05, Paris, 2005
- R. RAJUGAN, T. S. DILLON, E. CHANG, L. FENG: Modeling Views in the Layered View Model for XML Using UML; J. WEB. INFOR. SYST. 2 (2), JUNE 2006. Troubador pub. Ltd.
- FOR THE REMAINING BIBLIOGRAPHY SEE:
  C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, L. Tanca:A Data-oriented Survey of Context Models, SIGMOD Record, Dec. 2007. http://poseidon.elet.polimi.it/ca/