Using Relationship Patterns to Model Superimposed Information

Sudarshan Murthy, David Maier
Department of Computer Science, Portland State University
http://sparce.cs.pdx.edu
mailto:smurthy@cs.pdx.edu
The Superimposed System-Information Browser

- Allows a system (network) administrator to browse information about computers in a network
  - Applications installed and the modules they use
  - Updates applied
  - Errors recorded/reported
  - Application, system, and security events logged
  - User observations/comments
## The Browser

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Kind</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-Jul-04</td>
<td>06:07 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The McShield service was successfully</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:07 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The McShield service entered the stop...</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:06 PM</td>
<td>Sys...</td>
<td>USER32</td>
<td>The process winlogon.exe has initiated</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:06 PM</td>
<td>Sys...</td>
<td>Automatic Updates</td>
<td>The description for Event ID (21) in Sc</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:06 PM</td>
<td>Sys...</td>
<td>Automatic Updates</td>
<td>The description for Event ID (19) in Sc</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:04 PM</td>
<td>Sys...</td>
<td>Automatic Updates</td>
<td>The description for Event ID (17) in Sc</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>05:48 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>05:47 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>05:47 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>03:15 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>09:10 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>09:10 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>09:10 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:30 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>30-Jul-04</td>
<td>06:30 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>29-Jul-04</td>
<td>12:38 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>29-Jul-04</td>
<td>12:38 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>29-Jul-04</td>
<td>12:38 PM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The IMAPI CD-Burning COM Service se</td>
</tr>
<tr>
<td>28-Jul-04</td>
<td>08:16 AM</td>
<td>Sys...</td>
<td>Service Control Manager</td>
<td>The McShield service entered the runni</td>
</tr>
</tbody>
</table>
* All entities have key attribute ID (not shown); all relationships are many-many

A Conceptual Schema

- Observation
  - Date
  - Time
  - Text
  - User

- Event
  - Date
  - Time
  - Kind
  - Source
  - Description

- Computer
  - Name

- Error
  - Date
  - Time
  - Source
  - Description
  - Notes

- Module
  - Name

- Application
  - Name

- Update
  - Title
  - Description
  - Reason

- Applied on

- Occurs on

- Runs on

- Applies to

- Uses

Logged on

Relates to

Relates to

Involved

Applies to
Some structural variations exist, but information is neatly in a table

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Kind</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/30/2004</td>
<td>5:08:14 PM</td>
<td>Event Log</td>
<td>Infor Non6005 UN/C1</td>
<td>The Event log service was started.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:08:14 PM</td>
<td>Event Log</td>
<td>Infor Non6009 UN/C1</td>
<td>Microsoft (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Free.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:08:14 PM</td>
<td>Topip</td>
<td>Infor Non4201 UN/C1</td>
<td>The system detected that a network adapter CNet PRO200WFL PCI Fast Ethernet Adapter was connected to.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:07:25 PM</td>
<td>Event Log</td>
<td>Infor Non6006 UN/C1</td>
<td>The Event log service was stopped.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:07:04 PM</td>
<td>Service Control</td>
<td>Infor Non7035 NT/C1</td>
<td>The McShield service was successfully sent a stop control.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:07:02 PM</td>
<td>Service Control</td>
<td>Infor Non7036 UN/C1</td>
<td>The McShield service entered the stopped state.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:06:58 PM</td>
<td>USER32</td>
<td>Infor Non1074 NT/C1</td>
<td>The process winlogon.exe has initiated the restart of C1 for the following reason: No title for this reason could be found.</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:06:53 PM</td>
<td>Automatic Updates</td>
<td>Info -2</td>
<td>The description for Event ID (21) in Source (Automatic Updates) cannot be found. The local computer may not have the necessary registry information or message DLL files to display messages from a remote computer. You may be able to use the /AUXSOURCE=flag to retrieve this description; see Help and Support for details. The following information is part of the event: - Cumulative Security Update for Internet Explorer 6 Service Pack 1 (KB967801).</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:06:53 PM</td>
<td>Automatic Updates</td>
<td>Info -2</td>
<td>The description for Event ID (15) in Source (Automatic Updates) cannot be found. The local computer ma</td>
</tr>
<tr>
<td>7/30/2004</td>
<td>5:04:53 PM</td>
<td>Automatic Updates</td>
<td>Info -2</td>
<td>The description for Event ID (17) in Source (Automatic Updates) cannot be found. The local computer ma</td>
</tr>
</tbody>
</table>
### Error Reports

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/21/2003</td>
<td>15:44</td>
<td></td>
<td>Faulting application explorer.exe, version 6.0.2800.1106, faulting module ntdll.dll, 5.1.2600.1106, fault address 0x0002f727, 0x000a;</td>
</tr>
<tr>
<td>3/21/2003</td>
<td>15:44</td>
<td></td>
<td>Faulting application explorer.exe, version 6.0.2800.1106, faulting module msonsext.dll, 10.145.3810.0, fault address 0x000504c7, 0x000a;</td>
</tr>
</tbody>
</table>

Uniform structure, but mapping is not clean: Date and Time are both in Time field

---

26-Oct-05 Using Relationship Patterns to Model Superimposed Information
Data is heterogeneous and distributed: some data in XML, some in HTML

Structure varies: support URL not always defined, HTML page structure varies widely

[Update Table]

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data is heterogeneous and distributed: some data in XML, some</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structure varies: support URL not always defined, HTML page</td>
<td></td>
</tr>
<tr>
<td></td>
<td>structure varies widely</td>
<td></td>
</tr>
</tbody>
</table>

26-Oct-05  Using Relationship Patterns to Model Superimposed Information
Observations

• Heterogeneous data models and schemas
  – Event logs are in MS Excel spreadsheets, Error reports in MS Word documents

• Distributed sources
  – Master list of updates is on the LAN, support pages are on the web

• The various data are interconnected
  – *Outlook errors stopped after SP2 was applied*

• The conceptual schema hides the heterogeneity and distribution, yet allows us to navigate the interconnections
The Problem

- The conceptual schema hides *too much*
- It does not make explicit the presence of external entities (*base information*) and the references to those entities (*marks*)
  - One consequence: any logical schema generated is incomplete (with respect to representation of information referenced)
The Proposal

• Use a *relationship pattern language* to represent the use of marks
  - Identify and describe contexts for relationship patterns
  - Define schema-level and instance-level constraints
  - Fix syntax and semantics of relationship types
  - Describe consequences of relationships
Outline

• Motivation
• Some alternative solutions
• Overview of relationship patterns
• A relationship pattern language to represent the use of marks
• Conversion to logical model (relational model)
• Querying
• Summary
Model use of Mark as a Relationship

- Semantics of a relationship are mostly inferred from its name (and the definition of participating entities)
  - ‘Assign’ relates aircrafts and routes, but under what conditions should they be related?
- The traditional relationship does not completely capture the semantics of a mark
  - We need to distinguish between \textit{inter-layer} and \textit{intra-layer} relationships
ER Relationships Require Entities

- ER relationships are between entities, but sometimes an attribute carries a reference (e.g., Update.Title)
- Promoting attributes to entities, to show relationships, can cause entity proliferation (reduces comprehension)
  - The example schema has 12 such attributes
- Sometimes a group of attributes share a mark (e.g., Error.Date and Error.Time)
  - Can be hard to define a key for an entity created for a group of attributes
Attribute Value

• In ER, no dereferencing is involved in obtaining an attribute’s value, but obtaining a value from an attribute that uses a mark involves dereferencing
  - *E.g.,* Update.Title is the text excerpt of a mark

• Introducing a new domain such as ‘Mark’ does not suffice
  - We need to be able to distinguish between a value that is a mark and a value obtained using a mark
Supported Relationships

- Some relationships have support
  - An error applies to an application based on information in the details of the error report

- Traditional representation would use a relationship attribute
Superimposed Schematics*

• A superimposed schematic is an ER schema over base information

• One mark may be associated with an entity or a relationship

• Relationships cannot have attributes

• Introduces a Mark value type (?)

Our Approach

• Represent the use of a mark as a relationship

• We use relationship patterns to represent the use of marks
  – We define a relationship pattern language (a set of relationship patterns)

• No need for a ‘mark’ attribute or value type
  – That type can be added orthogonally
Relationship Patterns*

- A *relationship pattern* is an abstraction of *recurring* needs or problems when establishing relationships in a context; it can also be a *suggested* solution to the problems identified.
- A relationship pattern is similar to a software pattern, except it is focused on relationships.
- Like software patterns, inspired by the notion of patterns in architecture.

Example: The *Predicated* Relationship Pattern

\[ \text{<type>(<predicate>)} \]

- \text{<type>} is name of a relationship type;  
- \text{<predicate>} is a pre-condition for a relationship instance  

- \textit{E.g.,} An aircraft can be assigned to a route only if it can fly at least 25\% farther than the route’s distance

<table>
<thead>
<tr>
<th>Aircraft</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>Range</td>
<td>Distance</td>
</tr>
</tbody>
</table>

Assign \((\text{Range} > 1.25 \times \text{Distance})\)
Example: The *Computed* Relationship Pattern

<table>
<thead>
<tr>
<th>Aircraft</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>Range</td>
<td>Distance</td>
</tr>
</tbody>
</table>

Computed: <type>(<predicate>)

- Relationship instances are *computed* (not stored)
  - Traditionally, relationship instances are stored
- Relationship must not have attributes, or they must be computable
- Creates the *Computed* typespace
  - A typespace is a set of related types
Relationship Signatures

• A relationship pattern defines a syntax to create the three text parts of a relationship type: names of typespaces and types, role names, structure of cardinality constraints

• Each of these three parts is defined using a signature (formally a grammar)
  – *E.g.*, `<type>(<predicate>)` is a type signature

• The three signatures together are called the *relationship signature*
Why Use Relationship Patterns?

• Solve a kind of problems once
• Describe many relationship types at once
• Understand many relationship types at once
• Customize
  – Define how relationships are treated in various stages of the information life cycle
• Leverage known patterns
  – Following a pattern well-understood can ensure consistency and increase acceptance
Benefits when Representing Use of Marks

• Provide visual representation of the use of marks
• Any model element can be associated with marks (zero or more marks)
• Distinguish between a mark as a value and the use of a mark
• Provide a means to generate logical schema for superimposed and base information
  - Enables bi-level querying (over superimposed and base information, as if they are at the same level)
Representing the Use of Marks
Where can a Mark be?

• Entity
  – *E.g.*, Event

• Relationship
  – *E.g.*, ‘Applies to’

• Entity and relationship attribute
  – *E.g.*, Update.Title and AppliedOn.Date
Modeling Marks

• The *Mark* entity models a mark
  - The ID attribute uniquely identifies a mark; all marks support the function *resolve*
  - The use of a mark is shown as a relationship with this entity

• All *inter-layer* relationships are between a superimposed entity and the Mark entity
  - *Intra-layer* relationships are between entities in a single layer: superimposed layer or base layer
  - Our focus is on inter-layer relationships
The *Entity-Mark* Pattern

- The EMark typespace contains relationship types that associate entities with marks
- EventDetail associates an Event entity with a mark
- ‘Logged on’ is a traditional relationship type
Entity-Mark Details

• Type Signature

   \texttt{EMark:<type>}

• Constraints
  - Entity type and degree: Any superimposed entity type; any number of superimposed entity types
  - Cardinality: Any

• Semantics
  - Superimposed entities are associated with marks

• Consequences
  - Conversion to relational model presented later
The **Attribute-Mark Pattern**

- The AMark typespace contains relationship types that associate attributes with marks.
- ErrorDetails associates the Description attribute with a mark.
- ErrorTime associates attributes Date and Time with one mark.
Attribute-Mark Details

• Type Signature
  \[ \text{AMark} : \text{<type>} (a_1, a_2, \ldots a_n) \]

• Constraints
  - \(a_1, a_2, \ldots a_n \ (n > 0)\) are distinct attributes of a superimposed entity

• Semantics
  - All attributes specified are associated with the same mark (or same bag of marks if cardinality is greater than 1)
  - Associating an attribute with a mark does not mean its value is obtained using the mark
Combining AMark Relationship Types

- The AMarks typespace lets you “combine” many AMark relationship types that involve the same entity type (but imposes a common name, and cardinality constraints).
- The ‘Error’ relationship type associates the Date and Time attributes with one mark, and the Description attribute with one mark.
AMarks Details

• Type Signature

\[ \text{AMarks:<type>}(A_1, A_2, \ldots A_n) \]

• Constraints
  - \( A_1, A_2, \ldots A_n \ (n>0) \) are \textit{non-empty, disjoint} sub-sets of the attributes of a superimposed entity
  - Attribute sets may be indicated using braces or parentheses

• Semantics
  - Each \textit{set} of attributes is associated with \textit{one} mark (or a bag of marks)
Deriving Attribute Values from Marks

<table>
<thead>
<tr>
<th>Update</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>1 ID</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Reason</td>
<td></td>
</tr>
</tbody>
</table>

- An attribute might *always* derive its value from a mark’s context (e.g., excerpt)
- The VAMark and VAMarks typespaces define relationship types for this purpose
- UpdateDetail associates the value of each of the attribute Title, Description, and Reason with the context of a mark
VAMark Details

• Type Signature
  \[ \text{VAMark:<type>}(a_1, a_2, \ldots a_n) \]

• Constraints
  - \( a_1, a_2, \ldots a_n \ (n>0) \) are distinct attributes of a superimposed entity
  - Cardinality \textit{must} be 1 (single-valued attributes)

• Semantics
  - \textit{All} attributes specified are associated with \textit{one} mark, and their values are derived from that mark’s context

• Consequences: Requires casting/type checking
VAMarks Details

• Type Signature

\[ \text{VAMarks: \langle \text{type}\rangle(A_1, A_2, \ldots A_n) } \]

• Constraints

- \( A_1, A_2, \ldots A_n \) (\( n > 0 \)) are non-empty, disjoint sub-sets of the attributes of a superimposed entity
- Cardinality \textit{must} be 1

• Semantics

- Each \textit{set of attributes} is associated with \textit{one mark}
- Use of context is similar to that in the VAMark typespace
The *Relationship-Mark Pattern*

- Aggregate* the relationship to be associated with marks (called *supported relationship*)
- Add an RMark relationship with the aggregate
- The ‘AppliesTo’ relationship type is first aggregated. RMark:Application associates the aggregate with marks

* Ramakrishnan and Gehrke. Database Management Systems, 3rd Ed.
Avoiding Drawing Aggregates

- We draw a dotted line from the supported relationship (e.g., ‘Applies to’) to the Mark entity instead of drawing an aggregate entity
  - The dotted line clarifies that the degree of the supported relationship is unchanged
Relationship-Mark Details

• Type Signature

\[ \text{RMark} : <\text{type}> \]

• Constraints on the supported relationship
  - Can be inter-layer or intra-layer
  - Can be of any type, degree, and cardinality
  - Can have attributes

• Constraints on RMark relationship type
  - Always \textit{binary}
  - Can have attributes
* An update log stores details of applications of updates to computers

**Associating Relationship Attributes with Marks**

- The RAMark typespace contains relationship types that associate relationship attributes with marks
- UpdateLog* associates both attributes Date and Time with one mark
RAMark Details

• Type Signature
  \[ \text{RAMark:}\langle \text{type}\rangle(a_1, a_2, \ldots a_n) \]

• Constraints
  - \( a_1, a_2, \ldots a_n \) (\( n>0 \)) are distinct attributes of a superimposed entity

• Semantics
  - All attributes specified are associated with one mark (or a bag of marks)
  - Associating an attribute with a mark does not mean its value is obtained using the mark
RAMarks Details

• Type Signature

\[ \text{RAMarks: <type>}(A_1, A_2, \ldots A_n) \]

• Constraints
  - \( A_1, A_2, \ldots A_n \) (n>0) are non-empty, disjoint sub-sets of the attributes of a superimposed entity
  - Attribute sets may be indicated using braces or parentheses

• Semantics
  - Each set of attributes is associated with one mark (or a bag of marks)
* EMark, AMarks, VAMarks, and RAMark relationships are many-1; other relationships are many-many

**Revised Conceptual Schema**

[Diagram showing relationships between Observation, Computer, Event, Error, Application, Module, and their associated attributes and relationships.]

```
Observation
- Date
- Time
- Text
- User

Computer
- Name

Event
- Date
- Time
- Kind
- Source
- Description

Error
- Date
- Time
- Source
- Description
- Notes

Application
- Name

Module
- Name

REMARK: UpdateLog(Date, Time)

Other relationships...
```
Conversion to Relational Model
Converting the Mark Entity

• The Mark entity type is represented as a table with attributes such as
  - ID: Integer (key)
  - CreatedOn: Date
  - CreatedBy: String
  - CreateAt: String

• The attributes are derived from the SPARCE mark descriptor
Converting EMark Relationship Types

• Convert the relationship type and the superimposed entity type using the traditional procedure*

• Derive the name for the foreign-key attribute that references Mark.ID from the name of the relationship type.
  - E.g., EMark_EventDetail

Example EMark Conversion

CREATE TABLE Event

( ID Integer NOT NULL PRIMARY KEY,
  EDate Date, ETime Time,
  Kind CHAR(5),
  Source VARCHAR(25),
  Description VARCHAR(255),
  EMark_IncEventDetail Integer NOT NULL
  REFERENCES Mark(ID)
)

*Added ID attribute (for all relations). Altered names of attributes Date and Time
Converting AMark(s) Relationship Types

• AMark: Convert the relationship type and superimposed entity type using the traditional procedure
• AMarks: For each set of attributes in the parameters
  – Follow the procedure to convert AMark relationship types
Example AMarks Conversion

CREATE TABLE Error
( ID Integer NOT NULL PRIMARY KEY, EDate Date, ETime Time, AMark_Error_DT Integer NOT NULL REFERENCES Mark(ID), Source VARCHAR(25), Description VARCHAR(255), AMark_Error_Desc Integer NOT NULL REFERENCES Mark(ID), Notes VARCHAR(255) )
Converting VAMark Relationship Types*

- Follow the procedure to convert AMark relationship type
- Replace each attribute associated with a mark, with an integer attribute
  - The replacement attribute stores the ID of the context element that supplies the original attribute’s value
  - Alternative: remove the attribute, specify the context element ID in view definition (if value is always derived from the same context element)
- Define a view

*The procedure might not preserve key constraints if a key attribute is associated with a mark
Defining a View

• The schema of the view matches the entity’s
• For each attribute associated with a mark, embed call to the function `context`
  – The attribute that represents the associated mark supplies the mark ID
  – The attribute that represents the associated context element supplies the context element ID*
• We assume the view inserts a NULL value in case of a type mismatch (possible if function `context` returns an incompatible type)

*Alternatively, context element IDs can also be directly specified in the view definition.
Converting VAMarks Relationship Types

• For each set of attributes in the parameters
  – Follow the procedure to convert VAMark relationship types
Example VAMarks Conversion

CREATE TABLE  Stored_Update
( ID Integer NOT NULL PRIMARY KEY,
  VAMark_TitleCElm Integer,
  VAMark_Title Integer NOT NULL
       REFERENCES Mark(ID),
  VAMark_DescCElm Integer,
  VAMark_Desc Integer NOT NULL
       REFERENCES Mark(ID),
  VAMark_ReasonCElm Integer,
  VAMark_Reason Integer NOT NULL
       REFERENCES Mark(ID) )
Example View Definition

CREATE VIEW Update (ID, Title, Description, Reason) AS

SELECT
ID,
context(VAMark_Title, VAMark_TitleCElm),
context(VAMark_Desc, VAMark_DescCElm),
context(VAMark_Reason, VAMark_ReasonCElm)
FROM Stored_Update

• context is a user-defined function
Example Alternative VAMarks Conversion

CREATE TABLE Stored_Update
(   ID Integer NOT NULL PRIMARY KEY,
    VAMark_Title Integer NOT NULL
        REFERENCES Mark(ID),
    VAMark_Desc Integer NOT NULL
        REFERENCES Mark(ID),
    VAMark_Reason Integer NOT NULL
        REFERENCES Mark(ID)
)

VAMarks:UpdateDetail( (Title),
(Description), (Reason) )
Example Alternative View Definition

CREATE VIEW Update (ID, Title,
   Description, Reason) AS
SELECT ID,
   context(VAMark_Title, e1),
   context(VAMark_Desc, e2),
   context(VAMark_Reason, e3)
FROM Stored_Update

• e1, e2, e3 are IDs of context elements
Converting RMark Relationship Types

(1)*

• Convert the original relationship type and the related entity types using an appropriate procedure (the original relationship might not be traditional)

• To the table that captures the original relationship type
  - Add a foreign key attribute that references Mark.ID
  - Add attributes of the RMark relationship type

*Cardinality of the RMark relationship type is 1; cardinality of the original relationship type is immaterial
Converting RMark Relationship Types (Many)*

- Convert the original relationship type and the related entity types using an appropriate procedure
- Create a new table (derive name from the RMark relationship type). To the new table:
  - Add the key of the table that captures the original relationship type, and make it a foreign key
  - Add a foreign key attribute that references Mark.ID
  - Define primary key as set of foreign key attributes
  - Add attributes of the RMark relationship type

*Cardinality of the RMark relationship type is many
Example RMark (Many) Conversion

CREATE TABLE Stored_Update
( ID Integer..., PRIMARY KEY ID)
CREATE TABLE Application
( ID Integer..., PRIMARY KEY ID)
CREATE TABLE AppliesTo
( UID Integer..., AID Integer..., PRIMARY KEY
  (UID, AID))
CREATE TABLE RMark_Application
( UID Integer..., AID Integer...,
  RMarkID Integer
  REFERENCES Mark(ID),
  PRIMARY KEY (UID, AID, RMarkID))

* In the running example, Update information is stored in table Stored_Update
Converting RAMark Relationship Types (1)*

- Convert the original relationship type and the related entity types using an appropriate procedure
- To the table that captures the original relationship type
  - Add a foreign key attribute that references Mark.ID
  - Add attributes of the RAMark relationship type

*Cardinality of the RAMark relationship type is 1
Converting RAMark Relationship Types (Many)*

- Convert the original relationship type and the related entity types using an appropriate procedure
- Create a new table (derive name from the RAMark relationship type). To the new table:
  - Add the key of the table that captures the original relationship type, and make it a foreign key
  - Add a foreign key attribute that references Mark.ID
  - Define primary key as set of foreign key attributes
  - Add attributes of the RAMark relationship type

*Cardinality of the RAMark relationship type is many
Example RAMark (1) Conversion

CREATE TABLE Stored_Update
( ID Integer..., PRIMARY KEY ID)
CREATE TABLE Computer
( ID Integer..., PRIMARY KEY ID)
CREATE TABLE AppliedOn
( UID Integer..., AID Integer..., EDate As Date, ETime As Time,
 RAMark_UpdateLog Integer
 REFERENCES Mark(ID),
 PRIMARY KEY (UID, AID))
Using Views
When to use Views

• If an attribute *always* gets its value from the context of a mark

• When *live* base data is needed

• The VAMark and VAMarks typespaces automatically generate view definitions
  - We describe the use of views for “black belts”
Creating View Definitions

• Create a stored relation containing *only* the foreign key attributes that reference Mark.ID, and the attributes whose values are *not* derived from context of marks
  – Alternatively, replace an attribute that derives value from a mark’s context with an integer attribute that stores the context element ID

• Create a view over the stored relation with embedded calls to the function `context(a user-defined SQL function)` to compute values of attributes omitted from the stored relation
Example Stored Relation: Event

CREATE TABLE Stored_Event

( ID Integer NOT NULL PRIMARY KEY,
  Kind CHAR(5),
  EMark_EventDetail Integer NOT NULL REFERENCES Mark(ID)
)

*Application knowledge tells us that all but the ID and Kind attributes get their values from a mark’s context

Attributes EDate, ETime, Source, and Description are removed*

...
Example View Definition: Event*

CREATE VIEW Event (ID, Date, Time, Kind, Source, Description) AS

SELECT
  ID,
  context(EMark_EventDetail, e1),
  context(EMark_EventDetail, e2),
  Kind,
  context(EMark_EventDetail, e3),
  context(EMark_EventDetail, e4)
FROM Stored_Event

*e1, e2, e3, e4 are IDs of context elements
Example Stored Relation: Error

CREATE TABLE Stored_Error
(
  ID Integer NOT NULL PRIMARY KEY,
  Source VARCHAR(25),
  Notes VARCHAR(255),
  AMark_Error_DT Integer NOT NULL REFERENCES Mark(ID),
  AMark_Error_Desc Integer NOT NULL REFERENCES Mark(ID)
)

Attributes EDate, ETime, and Description are removed
Example View Definition: Error*

CREATE VIEW Error (ID, Date, Time, Source, Description, Notes) AS

SELECT
    ID,
    context(AMark_Error_DT, e1),
    context(AMark_Error_DT, e2),
    Source,
    context(AMark_Error_Desc, e3),
    Notes
FROM Stored_Error

*e1, e2, e3 are IDs of context elements
Querying
Bi-level Queries

• *Bi-level queries* can be written against the logical schema

• A query can freely use the function `context` with a mark ID and a context element ID
  - This function returns *live* data from the base layer (under normal circumstances)
  - Can assign the result of this function to an attribute
  - Can use function `excerpt` to retrieve text excerpt

• View definitions provide the best abstraction
Example Queries 1, 2

- Retrieve all update details
  SELECT * FROM Update

- Retrieve updates related to security
  SELECT * FROM Update
  WHERE Description LIKE 'Security%'

- Because Update is a view, values of attributes associated with mark are retrieved from the base layer when the view definition is executed
Example Query 3

• Retrieve all errors MS Word caused in the last week

```sql
SELECT * FROM Error
WHERE EDate BETWEEN CURRENT_DATE AND CURRENT_DATE - INTERVAL '6' DAY
AND Description LIKE '%Word.exe%'
```

• If `Error` is a view, the attributes date, time and description are retrieved from the base layer when the view definition is executed
Example Query 4

• Create a timeline of errors related to MS Word and MS Outlook

SELECT EDate, ETime, Description
FROM Error
WHERE Description LIKE '%word.exe%'
OR Description LIKE '%Outlook.exe%'
Sample Results 4

<table>
<thead>
<tr>
<th>EDate</th>
<th>ETTime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/26/2004</td>
<td>19:46</td>
<td>Hanging app...Outlook.EXE...</td>
</tr>
<tr>
<td>1/27/2004</td>
<td>20:04</td>
<td>Faulting app...winword.exe...</td>
</tr>
<tr>
<td>3/9/2004</td>
<td>16:38</td>
<td>Hanging app...winword.EXE</td>
</tr>
<tr>
<td>4/13/2004</td>
<td>10:11</td>
<td>Faulting app...Outlook.EXE...</td>
</tr>
<tr>
<td>4/23/2004</td>
<td>13:04</td>
<td>Hanging app...Outlook.EXE...</td>
</tr>
<tr>
<td>5/21/2004</td>
<td>9:39</td>
<td>Faulting app...winword.exe...</td>
</tr>
<tr>
<td>5/26/2004</td>
<td>14:05</td>
<td>Faulting app...winword.exe...</td>
</tr>
</tbody>
</table>
*Drawn using an XML transformation based on work of Nicolas Kruchten. Timeline is non-linear
Example Query 5

- Create a timeline of errors, along with the faulting application and module

```
SELECT EDate, ETime,
    SUBSTRING(Description SIMILAR "%" application ",", ESCAPE '\'),
    SUBSTRING(Description SIMILAR "%" module ",", ESCAPE '\')
FROM Error
```
## Sample Results 5

<table>
<thead>
<tr>
<th>EDate</th>
<th>ETime</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/26/2004</td>
<td>19:46</td>
<td>Outlook.EXE</td>
<td>hungapp</td>
</tr>
<tr>
<td>3/9/2004</td>
<td>16:38</td>
<td>winword.EXE</td>
<td>WINWORD.EXE</td>
</tr>
<tr>
<td>4/13/2004</td>
<td>10:11</td>
<td>Outlook.EXE</td>
<td>ntdll.dll</td>
</tr>
<tr>
<td>5/21/2004</td>
<td>9:39</td>
<td>winword.exe</td>
<td>winword.exe</td>
</tr>
<tr>
<td>5/26/2004</td>
<td>14:05</td>
<td>winword.exe</td>
<td>mso.dll</td>
</tr>
</tbody>
</table>
Timeline 5

Application and module information retrieved from context

Date and time information retrieved from context
Example Query 6

• What events related to Outlook are recorded after SP2 update was applied?

SELECT
    E.EDate, E.Time, E.Description
FROM Event E, Update U JOIN AppliedOn A
    ON U.ID=A.UID
WHERE U.Description LIKE '%SP 2%'
    AND E.EDate > A.EDate
    AND E.Description LIKE '%Outlook.exe%'
Summary

• Associating marks with entities, attributes, and relationships is a recurring need. That is, there are patterns involving use of marks.

• We have identified key aspects for patterns of using marks: contexts, constraints, syntax, semantics, and consequences.

• We have shown how to generate relational schema from a conceptual schema.

• We have demonstrated some bi-level queries.
References


• Murthy, Maier. A Framework for Relationship Pattern Languages