A Working Generic Canonical Schematic UIMS for an ODBMS

E. Essenius (corresponding), M. Sim, P. Kist, N. Simon
Bitbybit Information Systems
Kluyverweg 2a, Delft, The Netherlands
Phone: +31 15 2682569, Fax: +31 15 2682530
email: info@bitbybit-is.nl, URL: http://www.bitbybit-is.nl
W. Gerhardt
Faculty of Information Technology and Systems
Mekelweg 4, Technical University Delft, The Netherlands

Abstract
We demonstrate a visual User Interface Management System (UIMS) we built for our ODBMS, Perspective-DB [2]. The interface is generic because the presentation is based on an object model, it is canonical because the presentation of the object model is graphical; it is schematic because the database structure **and contents** are represented by the same constructs of the graphical model. The interface serves developers as well as professional and novice end-users. Iris is a commercial timetabling system built on Perspective-DB. We will demonstrate that the concepts of the UIMS support the complete timetabling process.

Keywords
ODBMS, UIMS, representation and modeling.

1 CONCEPTS OF THE CANONICAL SCHEMATIC PRESENTATION

1.1 Introduction

A database legend is a set of generic constructs of a comprehensive **graphical** object model. A database designer can apply the graphical object model, according to the object model rules, to produce a canonical object schema that can be interpreted by colleagues.
The problem: A static schematic presentation describes the application but does not enable manipulation. This is also true when the presentation describes dynamic aspects; the operations that object types can execute, for example. This means that the end-user cannot employ the application through the schema. At this point, even the advanced schematic UIs switch to tabular representation.
The solution: We extended the graphical object schema with the dynamical behavior of the application. The object schema can then play the role of UI for the application

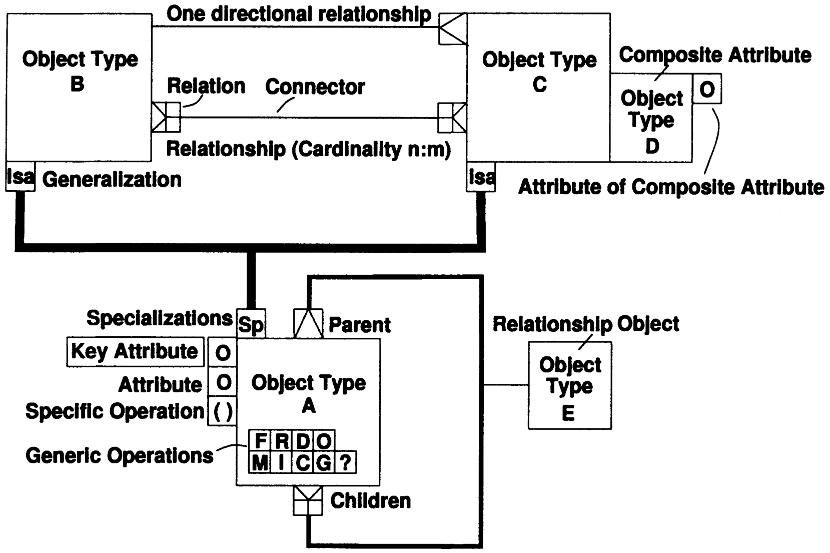


Figure 1 A compact legend of the Perspective-DB object model.

in employment mode.

The architecture: The UIMS is built on top of the ODBMS and can therefore utilize the high level object management available for applications such as notification, persistence and configuration. Because the UIMS and the application share the same ODBMS-API, the UIMS can easily access and manipulate the application, and monitor its status.

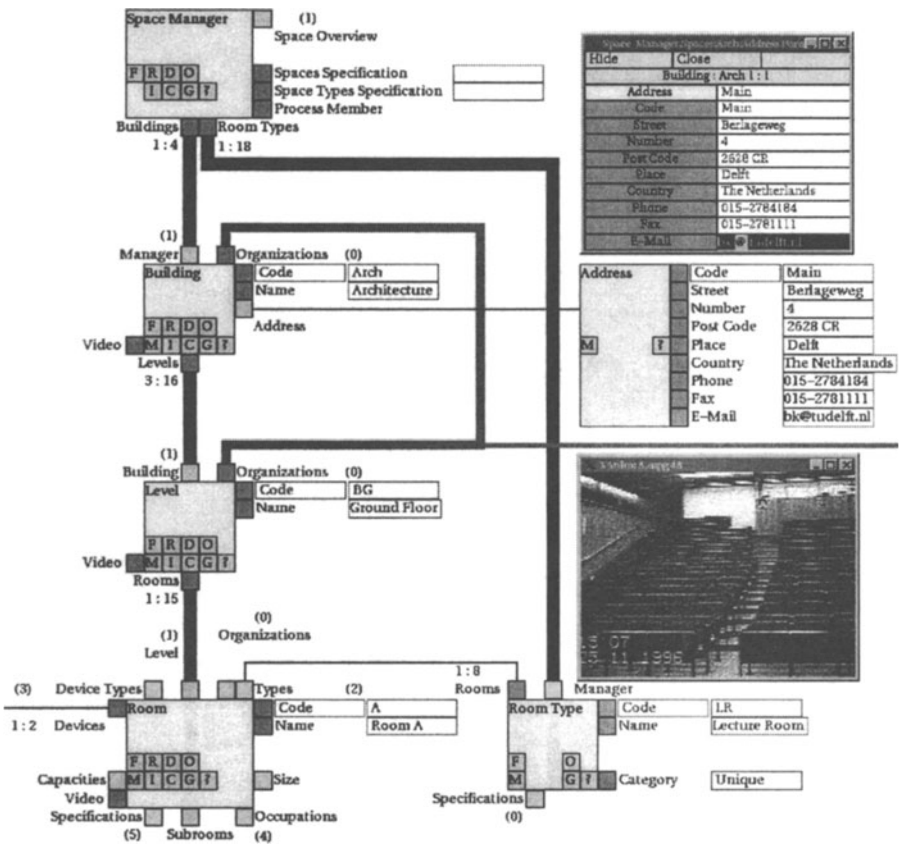
1.2 The Road Map for Database Structure Presentation

The compact legend explains the structural presentation:
Figure 1 shows a compact legend of the structural model which is similar to known models such as OMT [1] and Figure 2 shows a schema constructed with the model.

An object type has two kinds of *members*: properties and operations. A member is color-coded on screen and if relevant its name appears beside it. A *Property* is divided into *Attributes* and *Relations*.

Attribute constructs: A *Primitive* attribute is a literal, string etc (see Figure 2, 'Name' of 'Building'). An attribute can be a composite (see Figure 1). An attribute can be a multi-media type (see Figure 2, 'Video' of the object type 'Building'). A *Key* attribute is the unique identifier of an instance of a type with respect to its owner. (see Figure 2, the attribute 'Code' of 'Building').

Relationship constructs: A *Relationship* is composed of relations and *Connectors*. The cardinality is represented on screen by a number, supported by the semantics of the name i.e. singular or plural (see Figure 2, the relation 'Levels' of type 'Building'



Building: Arch 1 : 1

Address	Main
Code	Main
Street	Berlageweg
Number	4
Post Code	2628 CR
Place	Delft
Country	The Netherlands
Phone	015-2784184
Fax	015-2781111
E-Mail	bk@tudelft.nl

Code	Arch
Name	Architecture

Code	BG
Name	Ground Floor

Code	A
Name	Room A

Code	LR
Name	Lecture Room
Category	Unique

Figure 2 The space partition of the Iris system.

connects to the relation 'Building' of type 'Level'). A relationship is normally bi- but can also be uni-directional (see Figure 2, between 'Building' and 'Address').

An *Ownership* relationship upholds existence integrity and the connector is presented as a thick line. Only the *Owner* type can *Create* and *Destroy* objects of the *Owned* type. By convention, the owned type is placed below the owner, to project the structural hierarchy (see Figure 2, the relationship between 'Building' and 'Level').

A non-ownership relationship only allows *Inserting* or *Removing* objects, and this can be done from all participating relations. The non-owner relationship is presented as a normal line (see Figure 2, between 'Room' and 'Room Type'). It is possible to configure restrictions on which operations are permitted on its relations.

A *Relationship-object* is an attribute of a relationship. It is attached to the respective connector and is intended to model objects that only exist in the context of the relationship between object types (see Figure 1).

Operation constructs: An *operation* is an activity that an object type can execute

(see Figure 2, 'Process Member' of 'Space Manager'). A Specialization can inherit, overrule and disengage operations of its generalizations. A *Generic* operation is presented on the surface of an object type (see Figure 2, in object 'Space Manager'). A *Specific* operation executes an activity that is specific to an application (see Figure 2, 'Process Member' of object 'Space Manager').
Inheritance constructs: An *Inheritance* member defines the generalization- specialization hierarchy of object types and is presented as a very thick line (see Figure 1) and the specialization is positioned above the generalization by convention. The inheritance hierarchy is visible to developers and possibly administrators but is screened from end-users (see Figure 2, types 'Building' and 'Level' are actually specializations of type 'Space').

1.3 The Same Road Map for Database Contents Presentation

Object and Attribute: Presenting an instance 'on' its type is the key to interaction. By adding a *Value* to the member description, an object (type) represents the attribute values of its instance (see Figure 2, the 'Name' of 'Level' is 'Ground Floor' and its key 'Code' is 'BG'). The object now plays two roles, object type and *Object Instance*. The visible object instance is *active*. A special object type is the *Manager* of a *Partition*. A partition is a logical component of a large application (see Figure 2, 'Space Manager' manages the 'Space' partition). The manager instance is unique, is automatically created by the ODBMS, and cannot be destroyed. It is the natural entry point for the user to the instances in the partition. Instances can relate across partitions (see Figure 2, instance named 'Room A' relates to 2 'Devices' in the 'Device' partition.
Relation: By displaying the index beside the cardinality (e.g. 3:16) the schema shows which instance in the relation is active and the cardinality displays the actual number of instances in the relation (see Figure 2, relation 'Levels' of type 'Building' contains 16 levels and the third instance is active). The user clicks the relation to iterate backwards and forwards over the instances. Section 2 elaborates on how to *Navigate* the schema. Navigating within or across partitions is the same.
Operation: The user can click a generic operation of an active instance in order to **C**reate, **D**estroy, **I**nsert and **R**emove an instance; to **M**odify the instance's attributes; to **G**et a related instance based on an attribute value; to open a **F**orm (see Figure 2, the Form of the 'Address' of the 'Building' named 'Architecture') and an **O**verview of the related instances, and to get Help (?) documentation about the object and its members. The tabular (Form and Overview) and schematic presentations operate together, synchronized, as illustrated in Figure 2, or separately. Operations may have arguments and display the return (member) value. Section 2 elaborates on how to *Input* data.

The user can click a specific operation of an active instance in order to execute it. The user can click a multi-media object to get its presentation (see Figure 2, the image is the video of member 'Video' of the instance named 'Room A' of type

'Room').
Regulating the quantity of information on screen: The user can 'pop up' secondary objects on demand (see Figure 2, the instance with 'Code' 'Main' of type 'Address' pops up by clicking the member 'Address' of instance 'Architecture').

2 ELABORATION OF BEHAVIORAL CONCEPTS OF THE PRESENTATION

2.1 Enforcing Consistent Behavior

Providing consistent color coding: All interaction mechanisms must use the same color coding on all items to present the flow of user activities. An active item is gray, a selectable item is green, a selected item is blue, and an error condition is red.
Navigation path provides a consistent representation of the relationship: This path provides the necessary feedback to the user by visualizing the active relationships. When a user activates a relation in order to iterate, the relation forms a navigation path with the object type on the adjoining side of the relationship. This is visualized with the blue colored connector in between the corresponding types. As long as the navigation path is active, the source object type will keep the object type on the destination side consistent. When the active object is replaced, object types along the navigation path originating from relations of that type update the active object to reflect the actual situation in the database. Figure 2 shows the active relationships between 'Space Manager' and 'Room Types', and between 'Room Type' and 'Room'. When the user iterates the relation to room types, e.g. to get an instance 'Laboratory', the instance in 'Room' will be replaced with a room connected to 'Laboratory'.
Navigation across heterogenous relationships: In a heterogenous relationship, the navigation path is formed with green colored passive branches. The user can select one or more of these branches to become active (blue). For example, when the user navigates from an 'Organization' to the 'Space' partition (see Figure 2), the connectors from the joint to the 'Organizations' relation in 'Building', 'Level' and 'Room' are colored green. The user can now activate the branch to 'Building' to view the Buildings of that particular 'Organization'. When iterating a relation, only object types along active branches of the navigation path are updated.
The navigation path concept ensures an unambiguous representation: Each object type can only visualize instances from one active branch of any navigation path at a time. When a second branch to the same object type is activated the first will be deactivated automatically. Figure 2 shows that the relationship between 'Level' and 'Room' was deactivated, because the relation 'Rooms' of 'Room Type' was activated. Moreover, a navigation path always originates in one relation. When same connectors are used to bundle relationships, only one of the relations can be active at a time. When, for instance, the user has an active path from 'Level' to an 'Organization', and wants to activate the 'Organizations' relation in 'Building', the navigation

path originating from 'Level' is disabled. Moreover, the path will be disabled when the instance in the destination type is replaced, due to a 'Create' operation for example.

2.2 Input Protocol

Input is obtained from the schema: The input protocol serves two purposes which require similar input from the user; one to supply arguments for the invoked operations and the other to support formulation of queries. It superimposes the input behavior onto the schema, offering a uniform interface to the end-user for both input and output. The existing presentation concepts are applied, eliminating the need for a separate implementation.
Collection path of arguments: The arguments are collected along the *collection path* of the corresponding operation. The developer configures the collection path by indicating for each argument which elements can provide the information. When the user invokes an operation, the collection path is created for the first argument. When multiple elements can provide the required information, all possible providers turn green. The user selects an element, which turns blue while the others revert to their previous color. Then the user can provide the information either by entering a value, or navigating to an instance which has the required value. When finished, the next argument will be collected. The collection path ensures that all required arguments are collected in sequence. When activating the 'Modify' operation in 'Address' all attributes with the exception of key attribute 'Code' turn green. The user selects which attribute to change, and can then enter the new value. When all arguments are collected the operation will be executed.
Context dependent argument collection: Both the internal operations and the query mechanism require that arguments can be collected depending on earlier selections. When the user wants to create a new building, he selects the 'Create' operation in 'Space Manager'. Both the 'Buildings' and 'Room Types' relations turn green because they are both ownership relations. The user selects the relation to 'Building'. The navigation mechanism obtains the key attribute of the selected 'Building' and prompts the user to enter a new key value. The ODBMS then creates a 'Building' with the given key and 'Building' displays the new instance.

REFERENCES

[1] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. (1991) *Object Oriented Modeling and Design*, Prentice-Hall International, Inc, New York.

[2] Sim M., Gerhardt W., Kist P., and Simon N. (1994) The Specification of a Seamless O-O Programming Interface for CAD Tools, *TOOLS 13, Prentice Hall International*, UK.

3 BIOGRAPHY

E. Essenius, M. Sim, P. Kist and N. Simon: are key developers at Bitbybit Information Systems, founded in 1992 as a spin-off from five years R & D pioneering in object technology at the Delft University of Technology, The Netherlands. The authors continue to develop the ODBMS Perspective-DB and use it as the foundation to realize comprehensive information systems such as the IRIS system for timetabling, reservation and registration for academic organizations as demonstrated at the conference. The group's research is supported by SENTER (Dutch Ministry of Economic Affairs) and is carried-out in cooperation with the Database Systems section of the Delft University of Technology headed by Prof. W. Gerhardt. Further information about the authors can be found at http://www.bitbybit-is.nl.
W. Gerhardt: Waltraud was born on 23/1/49. Personal: has 2 sons, 2 grandchildren and 1 cat. Likes walking, gardening, and doing graphic design. Study: MSc in food-processing technology, specializing in sugar production; PhD in the mathematical modeling of the evaporation process for sugar production at the Humboldt University Berlin, GDR; Habilitation in data security in database systems at the Rostock University, GDR. Until '92 staff member and Assistant-Professor at Rostock University, Department of Computer Science. Since '92 chair holder of Database Systems, Delft University of Technology, Faculty of Information Technology and Systems. Special interests: object database technology, object modeling, multi-media databases and design automation.