Repository Design with an End-User Emphasis

John A. Miller
Walter D. Potter
Krys J. Kochut

Department of Computer Science & AI Programs
415 Graduate Studies Research Center
University of Georgia
Athens, GA 30602-7404

jam@cs.uga.edu

A. SITUATION:

- Repositories for Organizations NOT Committed to **CASE Technology**.

  Much of the software is **purchased**.

  Much of the software is **already developed**.

  Application developers are **not mandated** to use CASE.

    start-up costs

    prior negative results

    organizational inertia

B. DILEMMA:

- CASE Tools Provide the Most **Natural Means to Populate** a Repository

- Information from all phases of the **Development Lifecycle** can be obtained and stored:

  Requirements, Design, Implementation, Testing and Maintenance

C. APPROACH:

- Design the Repository from an **End-User Perspective** and Worry about Population Issues Later

- We use the term End-User in a **general sense** to mean any computer users whose primary mission is NOT the development of systems/applications.

- An End-User wishes to **use Information Technology** to accomplish some task.

- **Sophisticated** end-users who may do some **coding** or **configuring** of software are NOT excluded from this category.

- In most organizations this is the **largest and fastest growing category** of users.

- The **major components** of an information system that an end-user might see need to be analyzed. Special attention must be paid to their **interrelationships**.

D. WHAT AN END-USER SEES:

- Information_Resource
  - File
  - Database
  - Warehouse
- Application
- User
  - End_User
  - Application_Developer
  - Database_Designer
  - IS_Manager
- Business_Policy
  - Business_Goal
  - Business_Rule
- Model
  - Data_Model
  - Process_Model
- Tool
  - Development_Tool
  - Productivity_Tool
- System_Software
- Machine
- Network
- Business_Unit

E. HOW THEY ARE USED:

- An end-user will

  Read **document files**.

  Run **applications** that access **data files** and **data-bases**.

  Transfer software from one **machine** on a **network** to another.

  Run **productivity tools** such as Lotus 1-2-3.

  Use **development tools** such as compilers and linkers.

  May consult **models** to develop a big picture understanding.

  May install **systems software** such as DOS, Windows or Oracle.

- Of course, **complex tasks** will be done by **specialized staff members**.

F. SO WHAT'S THE REPOSITORY:

- The repository simply needs to **keep track of what end-users need to be aware of** to efficiently carry out their work.

- Not the actual data, but data about what data is available (e.g., Employee and Department relational tables). This is termed **meta-data**.

- Here is a **simplistic explanation:**

  Take a **Database Data Dictionary** and a **CASE Encyclopedia** and **merge** them together.

  Now **generalize** so it works with a variety of DBMS's and CASE Tools.

  See what important information is **left out** (probably all sorts of files) and include that.

- From an End-User perspective, the CASE Encyclopedia side needs **adjustment**. All of this important information should not be thrown away.

G. FUNDAMENTAL DESIGN DECISIONS:

- Here lies the fundamental **design decisions**.

- How much and in what form should **typical CASE Encyclopedia information** be stored in a Repository servicing an organization NOT using CASE.

- Also, what **replacement mechanisms** should be used for **populating** the repository with this information.

- This was our **mission** in designing a prototype repository for the **Westinghouse Savannah River Company**.

H. REPOSITORY INFORMATION MODEL (RIM):

- The prototype's RIM was modeled using **Extended Entity- Relationship (EER) Diagrams**.

- These were created using the **Oracle CASE Designer**.

- Extensions:

  Specialized (Subentity) Types -- Oracle

  Aggregation Types -- custom extension

- In the future, an **Object-Oriented Model** would be preferred.

I. BUT FIRST SOME DEFINITIONS:

- A Database can be thought of as having two basic levels: the **Instance Level** and the **Schema Level**.

- Abstractly, one can think of a **Repository as Two Databases (DB's)**.

    One DB describes what's in an information system (in our case what end-users need to be aware of).

    The **Other DB** describes what's in the first database.

- A nice way to get the **big picture** is to imagine an **operational database** which is described by the **first repository database** which is, in turn, described by the **second repository database**.

- There is a **correspondence** or **interlocking** of these three databases: Information concerning the schema of a database is stored as instances in the immediately higher level database.

Level-Pair 1	Level-Pair 2	Level-Pair 3
		Repo. DB 2 Schema
	Repo. DB 1 Schema	Repo. DB 2 Instances
Oper. DB Schema	Repo. DB 1 Instances	
Oper. DB Instances		

Table 1: Three Level-Pairs.

J. FOUR LEVEL ARCHITECTURE:

- The interlocking produces a **four-level** architecture.
- Various **standards** have different names for each level.

general	IRDS	PCTE	CDIF
meta-meta-model	Fundamental	Meta Schema	Meta Meta
meta-model	IRD Definition	SDS	Meta
meta-data	IRD	Object	Model
data	Application	(undefined)	Application

Table 2: Standard Terminology for Levels.

- The **RIM** is the **model for Repo. DB 1's Schema**. A **Core RIM** is supplied by the vendors.

K. SO WHAT IS TO DESIGN:

- A repository should be **extensible**, especially the RIM. Furthermore, an organization does not have to use the entire Core Model just because it's provided.

- Most of the remainder of the talk will outline a **Design of a RIM for a Prototype Repository** emphasizing the needs of the **End-User Community**.

L. EER DIAGRAMS:

- **Diagrams** produced by the Oracle CASE Tool

Entity Types	-- Box
Subentity Types	-- Box within a Box
Entity Aggregates	-- Box with AE in corner
Relationship Types	-- Line (crow's feet -> many)
Attributes	-- not shown

- **Issues** to consider:

  How **far to drill down** on the development side?

  What kind of **linkage** should exist between major components?

  Should the components be controlled by the repository (**active**) or simply described (**passive**)?

Figure 1

Figure 2

Figure 3

Figure 4

Figure 5

Figure 6

Figure 7

Figure 8

Figure 9

M. POPULATING THE REPOSITORY:

- **Exportation from Integrated Tools**

  e.g., from CASE Tools

- **Extraction from Meta-Data Sources**

  e.g., from Oracle Data Dictionary, PowerBuilder pibble files

- **Scanning Schema/Source Code**

  e.g., from Oracle .sql files, C++ .h files, /etc files

- **Reverse Engineering**

  e.g., an EER Model from Oracle Data Dictionary, an OMT Model from C++ .h files

- **Resource Tracking**

  e.g., Apsylog, Inc’s PC Galaxy is a resource tracking utility

- **Direct Manual Entry**

  e.g., from printed documents

N. REPOSITORY SERVICES:

- **Configuration Management**

  Maintain connections and dependencies between modules. Perform automatic or assisted builds.

- **Version Management**

  Keep track of version histories. Assist users in finding appropriate versions.

  Version Control

  In an active repository, control of some versionable objects may be given to the repository (e.g., repository can control creation of software modules, as opposed to having no control or using SCCS).

  May use check-in, check-out

- **Impact Analysis**

  Assess the impact of a change on other software components as well as the user community.

- **Change Notification**

  Notify users of changes that may affect them using, for example, flags, initial messages or e-mail.

- **User Authorization**

  Access to Repository Proper

  The repository would control the access to the information it stores (e.g., meta-data concerning the organization's operational databases).

  Access to Information External to Repository

  Access to information referenced by the repository may be controlled by the repository, but it is typically more efficient to allow some other system to do the control (e.g., operational database's DBMS).

- **Tracking and Auditing**

  Maintain a sufficient **audit trail**.

  Modification Tracking

  Record all important changes / modifications, esp. to database schema and software modules. Keep track of who, why and when. Changes to databases are recorded in their logs.

  Usage Analysis

  Various schemes can be applied to record information about the usage of resources (e.g., database tables, applications).

- **Type Management/Transformation**

  Data type/structure definitions that are shared by multiple software modules or database schema should be stored in the repository. Repository should dispense transformed (to match a particular language syntax) copies to users on demand.

- **Subscription Service**

  The repository can maintain a list of subscribers for each report produced periodically by the organization.

O. A FEW FINAL THOUGHTS:

- **Technological Foundations**

  Relational Databases
  Object-Oriented Databases
  Multidatabases
  Persistent Storage Manager
  Distributed Object Manager (CORBA or Cairo)
  All of the Above

- **Interfaces**

  Query Tools
  Browsers
  CASE Tools

- **Some Bumps in the Road**

  A *Monolithic* Repository vs. More *Specialized* Repositories

  If the latter wins, *Integration Techniques* like those used for Multidatabases will be called for.

  A *harder problem* since a repository provides more services than a database. Depending on usage, it may be *easier* because of relaxed time constraints

  A completely active repository would require complete *interoperability* between components in the corporate information system.