Repository System Engineering

Philip A. Bernstein
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
e-mail: philbe@microsoft.com

Repositories manage metadata. Metadata describes complex artifacts that are the subject of formal design activities, such as business processes, application interfaces, database (DB) schemas, engineering drawings, software configurations, and document libraries. Demand for them is growing, fueled by enterprise re-engineering, integrated CASE, data warehouse, and management systems for networks, computer systems, information resources, documents, web sites, etc. It's hard to measure product revenue, because repositories are often embedded in other products, but it's arguably already a billion dollar per year business. It's likely to get a lot bigger.

At the core of a repository system is a *repository manager* — a generic database application that manages metadata. It supports functions for managing changes in the structure of objects: types, relationships, versions and configurations. But a repository system is much more than a repository manager. The system must include a rich tool set and an object model (called an *information model*, in repository-speak). It's the tools that make the system useful to people. By itself, a repository manager is pretty useless. Thus, to understand repository technology, it's essential to understand this system context.

A repository system includes generic tools that are useful in all application contexts, such as graphical and hierarchical browsers, configuration managers, scripting languages, and data translators. It may also include high-level engine functions, such as type-specific merging, impact analysis, and rebinding algorithms. A system also needs application-specific tool sets, which can span a wide range of activities: application analysis, design and development; document management; system and network management; etc.

Much of the work in engineering a repository system is in integrating application-specific tools with the repository manager. One needs to identify the objects that tools need to put in the repository to support tool features. For example, to navigate between objects on the screen, the repository needs to store relationships between the objects. There needs to be an information model that covers objects that multiple tools want to share. Often, "control glue" is needed to allow the tools, tool shell, and repository objects to call each other in both a tool-driven and event-driven way. Also tools must cross-post updates to tool-private repositories, a common legacy of pre-repository days.

Although the repository engine is not the whole system, it is certainly a key element. Ideally, it is an object-relational DB system, to support navigational and query access, along with functions for managing change. It supports extensible types, augmenting the underlying object model's types with information about repository behavior. It has functions to define new types and to extend type and class definitions, e.g. to change the type definition of existing objects.

The engine supports relationships between objects. It enforces relationship integrity. Most fancier relationship semantics are variations on operation propagation, where operations on an object are propagated to related objects.

Repository version operations are not much different than for file-based versioning: create, freeze, branch and merge. However, their semantics and implementation interacts strongly with operations on types and relationships. For example, updating a type definition amounts to creating a new version of the type. Version behavior is supported using successor relationships between objects. And creating a new version opens the question of what to do about relationships on its predecessor: ignore them, copy them, ...

A configuration is a versioned object with relationships to its contents. It can be a scope for names, relationship traversal, and security. It supports long transactions via checkout/checkin. Its implementation must be optimized for versioning and for certain transitive closure calculations.

Repositories are an important generic DB application, one that would benefit from more DB research attention.

Tutorial Outline

- Repository product and system features
- Examples - CASE, data warehouse, information mgmt
- How to integrate tools with a repository
- Engine architecture, database structure, performance
- Types, Objects, and Extensibility
- Relationships - operation propagation, binding models
- Versions and configuration - mechanisms, usage models, example systems

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50