WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

BUILDING FLEXIBLE, EXTENSIBLE TOOLS FOR METADATABASE INTEGRATION

Michael Siegel
Arnon Rosenthal

November 1991

WP # 3357-91-MSA
WP # CIS-91-11

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

BUILDING FLEXIBLE, EXTENSIBLE TOOLS FOR METADATABASE INTEGRATION

Michael Siegel
Arnon Rosenthal

November 1991

WP # 3357-91-MSA
WP # CIS-91-11

Building Flexible, Extensible Tools for Metadatabase Integration

Arnon Rosenthal
MITRE Corporation
arnie@mitre.org

Michael Siegel
Sloan School of Management
Massachusetts Institute of Technology
msiegel@sloan.mit.edu

1 Introduction

Applications that span multiple existing files and databases need an integrated description of the information that they access (i.e., an integrated metadatabase). As the number of cooperating systems increases, the development and maintenance of this integrated description becomes extremely burdensome for administrators. Large amounts of descriptive information must be integrated and much data modeling skill are required.

Generating and maintaining the relationships among the component schemas can become an administrative nightmare, especially in federations of autonomous data sources each of which is free to make changes to its metadatabase. As described in the component metadatabases, data from these sources may differ in representation (e.g., data models), structure (e.g., data formats, database schema designs) or in semantics (e.g., differing definitions of GNP). Therefore, developing and maintaining an integrated view of even a few autonomously operated sources would be a difficult task, and thus requires an automated *metadatabase integration tool.* Such a tool must assist in developing an understanding of the semantic connections between metadatabases and from this understanding, produce a combined metadatabase, including mappings between the combined database and the components databases[1].

The literature on metadatabase integration consists mostly of techniques identifying related information in the metadatabases being integrated and techniques for resolving differences between the two metadatabases' treatments. Most of the published work deals with schema information only, in the relational model or for various extended entity-relationship (*EER*) models. Even papers that contain system overviews (e.g., [BLN86]), give little attention to pragmatic issues that we consider crucial – flexible usage, and integrator customization and extension.

Our goal is to justify and call attention to the requirements, and to provide a framework for research on metadatabase integration tools. We envision a tool that supports: *scheduling*

[1]The term *metadatabase integration tool* is used to emphasize that the users' problem (and hence the tool's task) is broader than just integrating schemas – the tool should also handle other types of descriptive information. For example, attributes may be annotated with security level, audit priority, and several categories of textual comments[McC84].

flexibility, to allow integration to be done in different stages and along different levels of expertise; *metadatabase evolution*, to allow incremental development and evolution of the component metadatabases and the organization's metadatabase; *model extensibility*, to capture descriptive information that is not standardized across the industry; and *method extensibility*, to add new or site-specific integration methods. Finally, despite all the flexible orderings and extensions, the integrator should make minimize the requirements for user interaction.

Section 2 describes unmet pragmatic requirements on integrator tools, and in Section 2.1 we illustrate them with a sample session. An architecture that can meet these requirements is described in Section 3. Our work is intended to be model independent, e.g., to apply to integrating pairs of relational databases, pairs of EER schemas, or pairs of object-oriented schemas. Finally, Section 4 presents our conclusions.

2 Pragmatic Requirements on Metadatabase Integration

In this section we present a sample metadatabase integration session to demonstrate the requirements for a metadatabase integration tool. Then, we describe the importance of flexibility and extensibility in metadabase integration. In Section 3 we propose an architecture to address these requirements.

2.1 A Sample Session

This sample integration session illustrates scheduling flexibility and the extensibility that users need and our architecture supports. It also illustrates the use of small, concrete questions for eliciting interschema relationships from users. It is not intended to illustrate a sophisticated collection of rules – our interest is in the framework and the style of interaction. For simplicity, the sample session integrates relations rather than entities and relationships.

Consider the problem when two organization with overlapping activities decide to merge their information systems. The first stage of this effort is to provide a unified view of the two companies' data while continuing autonomous operation of the component databases. The information system integration project begins with a pilot project to integrate personnel data. Ms. Smith of Human Resources is the expert assigned to work with the Information Technology (IT) group. The project will use a vendor-supplied integrator that deals with structural integration issues in relational databases.

The IT group helps her develop a user profile for their effort. The default profile for a nontechnical user is breadth first (i.e., completing a higher level before considering details) and excludes rules that deal with mathematical dependencies and with versioning. The group decides to make one change to the default – as soon as a combinable pair of relations is identified, details of its integration will be worked out before proceeding to other relations.

Ms. Smith identifies the relations in each database that deal with personnel, and the integrator is instructed to restrict its attention to these relations. She then tells the integrator to begin execution.

A heuristic rule suggests that the relations, Database1.EMP and Database2.EMPL, shown in Figure 1 seem likely to be mergeable. Ms. Smith agrees with the suggestion, she

Company 1 - Database1
Relation **EMP**
Emp# - (###-##-####)
Name - Char(40)
Annual_Salary - Integer
Age - Integer
... etc. ...

Company 2 - Database2
Relation **EMPL**
E# - (###-##-####)
EName - Char(36)
Weekly_Pay - Integer
Date-of-Birth - Date
... etc. ...

Figure 1: Two Relations from the Sample Databases

is asked what the concept should be called in the combined schema. Instead of choosing one of the suggestions EMP and EMPL, Ms. Smith types EMPLOYEE.

Ms. Smith tells the system that the two E# attributes (i.e., Emp# and E#) are mergeable, and that Name and EName are mergeable. She accepts the default names for the merged attributes, E# and EName. According to her schedule, it is not yet time for her to deal with datatypes. However, the system knows that its rule for combining datatypes requires no human intervention when there is no conflict (i.e., E#, Emp#). In order to maximize progress toward a combined schema, the system executes this noninteractive rule.

Next, using a local thesaurus of Personnel jargon, the integrator guesses that Annual_Salary and Weekly_Pay may be mergeable. When Ms. Smith replies yes, the integrator asks for a name for the combined attribute, and a way to compute the proper value. She tells the integrator to defer this issue until later. Part way through the attribute list, Ms. Smith decides that her assistant is better able to handle definitions of EMPLOYEE attributes. When the system next requests information from her, she calls up a control screen and tells the system to proceed to the next type of descriptive information.

The integrator now gathers information necessary to determine the constraints on the combined relation and the mappings from the component relations to the combined relation. The number of possibilities can be substantial, and the user is nontechnical, so rather than present possible combined schemas and mappings, the integrator asks some simple questions (Ms. Smith's answers are bold and underlined):

- If records from EMP and EMPL have the same key value E#, must they refer to the same EMPLOYEE? (**Yes**, *No*, *Defer*)
- Can records from EMP and EMPL with different E# values refer to the same EMPLOYEE? (*Yes*, **No**, *Defer*)
- Can the same EMPLOYEE be in both relations EMP and EMPL? (**Yes**, *No*, *Defer*)

A set of rules in the integrator now infers that records with the same E# should be combined, that EMPLOYEE is the *outer*join of EMP and EMPL, and that E# is a key of EMPLOYEE. The results of attribute integration are used to specify the target list of the outerjoin. Answers to all the questions used to resolved this schema are retained.

The team continues filling in details of EMPLOYEE, and then of other relations. Suddenly, the auditors insist that the combined metadatabase be extended to include access controls. The component metadatabases include access lists on each attribute, but the vendor-supplied integrator does not handle this information. A member of the team writes a new rule that helps a user to merge access lists interactively; these rules are to be applied whenever a pair of attributes are merged. The programmer registers the new rule with the integrator, and it is immediately applicable.

After lunch, one member decides to generate a prototype combined metadatabase. Many of the rules (e.g., for datatype integration) can determine a confidence level for their default action; when the integrator runs with low required confidence, these rules will make most decisions without user interaction. The prototype is generated rapidly. Meanwhile, work proceeds on the accurate schema, and discussions begin about an enterprise conceptual model. The integrated metadatabase, though not complete, may be used to develop applications that require data from portions of both databases.

From this scenario, we conclude that several specific capabilities ought to be included in the integrator's skeleton: redefinition of the scope for the immediate integration effort; priority to fully-automated rules whenever all necessary inputs are available; deferral of inconvenient questions; redefining the set of active rules; simple, user-friendly questions whose answers eventually permit inference of more complicated relationships; reactivation of deferred questions; avoidance of redundant questions; adding new rules to cope with new kinds of metadata; and fast prototyping.

2.2 The Need for Flexibility and Evolution

The scenario identified two basic properties that an integrator ought to have, and that the current research literature ignores – flexibility and and extensibility. We now summarize the desired capabilities:

1. *Scheduling Flexibility* - The integrator must allow its user to control the order of task execution, in order to adapt to project goals or availability of resources. For example, the user must be allowed to choose the initial goals – a broad enterprise model that presents major entities, relationships, and attributes for the whole organization, or alternatively a narrow but complete pilot project (e.g., integrating inventory information from two corporations that are being merged). Hybrid goals also make sense, such as detailed integration of inventory information, plus a top-level integration for related areas such as order entry.

   A large integration task requires many kinds of expertise (e.g., about CAD techniques, design management, manufacturing, and security). Questions may need to be deferred until the expert is available. When experts are interviewed, the integrator should concentrate on questions relevant to their expertise. Also, spontaneously-offered information should be captured, even if the integration methodology relegates it to another stage.

2. *Extensibility by User Sites* - New integration techniques appear frequently in the research literature and may be implemented by integrator vendors, but these are not the

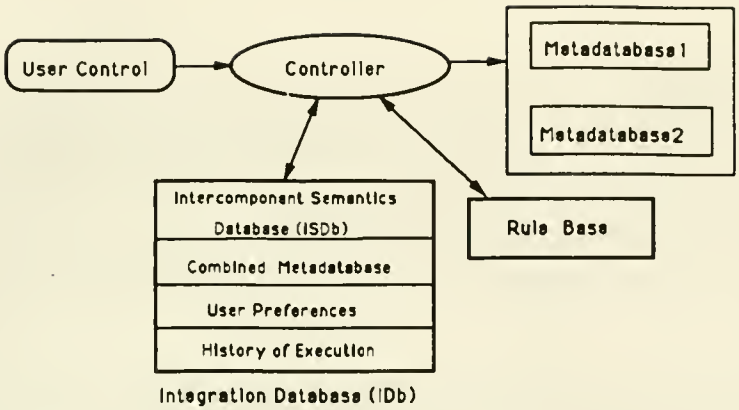

Figure 2: Proposed Integrator Architecture

only source of new technology. Sites may need to add rules that exploit local naming conventions, and will certainly need to develop their own rules to handle locally-defined kinds of metadata (e.g., accuracy, versioning policy, timeliness, charge-for-use, person-responsible, and myriad categories of textual comments). When adding these rules to purchased integrators, user sites do not want to learn the integrator's internals, and in fact may have no access to them.

3 Proposed Architecture

Integrators built with conventional coding techniques will find it very difficult to provide the flexibility and extensibility that were illustrated in the sample session. A rule-based system seems preferable. This section explores how a generic rule-based architecture needs to be refined and customized to support such new-generation integrators. We do not propose detailed designs and algorithms. Rather, we describe how the global requirements impact each component of a simple rule-based architecture.

The proposed architecture has four major modules described in Sections 3.1-3.3 and are shown in Figure 2. The arrows describe the flow of information. The architecture applies to an integrator that handles databases in any single model, and also permits extensions to the model.

3.1 The Integration Database (IDb)

The Integration Database provides stable, transaction-controlled storage for all information that needs to survive session boundaries, including all information gathered by rules, the user's preferences for the sequence of rule execution, and a history of rule executions (Section 3.3).

Each item of information generated by a rule is stored in the the Intercomponent Semantics Database (ISDb). The *descriptive* portion of the ISDb captures the "real-world" semantic relationships between the component databases. Typically there is only one "correct" answer for this information. The *prescriptive* portion captures design choices for the

combined metadatabase. In general, we prefer that the rules obtain and the ISDb store information in small, concrete increments, as illustrated by the scenario's questions about matching keys (E#). Such small questions tend to be more understandable by users, and require less rollback if a decision is to be changed.

The IDb includes information on object mergeability, intercomponent semantic relationships, and the combined schema.

3.2 Rules

Rules are the unified mechanism for expressing all the integrator's work, used for both high-level decisions (e.g., which relations shall be merged), and low-level decisions (e.g., how to resolve conflicts between specifications of String(9) or Integer for Part#s). The unified treatment facilitates tracking and explaining the impact of evolution in the component databases or the interschema semantics.

In many rule languages, one assumes that each predicate in a rule is inexpensive to evaluate. A schema integrator, though, includes many rules that require user input. Such input is normally more costly than any fully-automated rule, so the format from rules is chosen to minimize the number of interactions and other expensive operations.

A rule has three parts: a precondition, a body, and an action. The description below is an attempt to allow for performance optimizations (e.g., possible early evaluation of some predicates), while minimizing the interactions requested from the user. Note also that execution of the entire set of rules can take days, and it is possible to introduce information that conflicts with previous decisions, and for the user to edit the ISDb directly, possibly invalidating the results of early evaluation of predicate conjuncts.

The precondition is a conjunct of Boolean terms (with the structure visible to aid in performance optimization). Furthermore, the precondition should normally include only terms whose evaluation is fully automatic. The system is permitted to evaluate predicates in the precondition repeatedly, as conditions change. The body is evaluated only after the entire precondition becomes true for some binding[2]. Normally (i.e., if not deferred or restarted) the body will be evaluated just once for each rule binding. Costly operations should appear in the body rather than the precondition. The action is arbitrary; only it can modify the ISDb.

Rule format is illustrated below, for two simple rules. The bodies have been simplified to remove user-interactions that might otherwise be present. The outcomes insert an assertion about mergeability of the two attributes, or about the security level of the combined attribute.

> **Mergeable_attribs(A1, A2)** /* tells whether A1 and A2 should be combined
>
> Precondition: (The relation containing A1 is mergeable with the relation containing A2)
>
> Body: {If Name(A1) = Name(A2) then IDb_Insert(Mergeable(A1,A2))}

[2] A rule binding is a set of variable values that make the precondition True

Combine_Security_Levels(S1,S2) /* determines how the security levels should be integrated for two attributes that are being combined

Precondition: Si is the security property of attribute Ai from component database i, and A1,A2 are to be merged. A12 denotes the resulting attribute.

Body: {IDb_Insert(Security(A12)= maximum(S1,S2))}

3.3 User Control of the Integration Process

Conventional rule-based applications often run autonomously, or else interactively over seconds or minutes. Schema integration lasts for days (i.e., or may continuously evolve), during which the metadatabases may be changed due to external events or to edits by the user rather than by the action of a particular rule. As illustrated in the scenario, the user therefore needs to control the order of tasks. Control can be direct or indirect.

Users can obtain direct control in three ways: i) the controller can be asked to return control periodically or upon reaching certain points (analogous to a debugger stepping through or reaching a breakpoint); ii) an interrupt can abort the currently-executing rule; iii) whenever a user supplies input requested by a rule, the interface allows the user to take control actions.

The user has two basic choices after obtaining control. First, he or she can select a set of candidate rule bindings, and cause them to be invoked immediately. Second, the user can browse and edit all information in the IDb (i.e., subject to authorization).

A controller can consider several kinds of control information in order to choose the next rule binding to be executed. Here we present three examples of the types of control that are important in the integration process – *scoping, deferral, and directionality.* All were illustrated in the scenario. **Scopes** are views over the rule base and database. Rather than directly execute all stored rules over all information in the IDb, a scope provides a subset of each. Users can request execution within a scope (e.g., Ms. Smith's scope was *personnel*) that is more targeted to their interests and a smaller set of rules (e.g., excluding rules that deal with technical details of datatypes and security).

The integrator must be able to **defer** a rule bindings for which the user is not prepared to furnish the answer (e.g., Ms. Smith chose to defer interactions involving datatypes). The IDb includes a history of executed rule bindings, some of which are marked *executed – deferred.* Bindings to be reactivated may be selected by special commands or by an ordinary database query.

Finally, the user should be able to influence the **direction** of integration. For example, for hierarchical metaschemas the user may choose to go "down". This would mean to next apply candidate bindings associated with the components of a metatype, e.g., after determining that two relations can be merged, obtain information about the first child in the metaschema (e.g., Relation_name, Attribute_List, Constraint_List). Another choice would be "same_rule" where the user chooses to apply the same rule again but with different bindings.

3.4 Controller and Control Strategies

The controller determines the next rule binding to execute (one at a time). It is a key variable in the integrator's usability, but to date the control process has received little research attention in the database community.

Any rule binding whose precondition is True and which is not recorded as "executed" is called a *candidate* for execution. A controller must invoke only candidate rules, and not terminate until or candidates are exhausted, or the user issues an *exit* command. The controller stores the history of executed rule bindings, with return codes and other status information. The history is part of the IDb, and may be referenced by rules, and edited by built-in commands. Ideally, the controller will be built as an enhancement of a general-purpose rule engine, but it is not clear whether off-the-shelf rule systems allow their controllers to be modified in this way. An additional responsibility of the controller is to use indexing, eager evaluation, and similar techniques to minimize the user's time waiting for the next interaction.

4 Summary

We have identified unmet pragmatic requirements of long, interactive design processes such as metadatabase integration. We described some ways to adapt a generic rule-based architecture to meet those needs. Our design was quite preliminary, but points the way for future research.

The proposed architecture helps an integrator satisfy the goals of flexibility by allowing integration tasks (i.e., rule bindings) to be invoked in arbitrary order as long as each precondition is satisfied. Several mechanisms were proposed for influencing the choice of next rule to be invoked. Users can invoke sets of candidates, explicitly. The scope and direction of integration is adjustable at any time. Users can freely defer tasks (i.e., rule bindings), and have great flexibility in determining which deferred bindings should be reactivated. The proposed integrator provides for extensibility by using a rule-based approach to define all integration techniques. New techniques can be freely added to the rule base, to allow users to include techniques tailored to local conditions. The integrator will invoke these new rules when their preconditions are satisfied. This mechanism handles both the results of new research, and locally-developed rules to handle local kinds of metadata.

In a future database system, the metadatabase integrator will be just one of many tools. It will depend on other tools that help with evolution of multilevel schema structures, and will share knowledge with other tools that deal with data semantics. There is a substantial functional overlap between schema integrators, intelligent query-formulation assistants [KBH89,KN89], and intelligent assistants that reconcile database contents to a user view [SM91]: Much of the knowledge that they use is similar (e.g., about identifying similar concepts between two views, or about necessary conversions). Also, all three tools generate mappings from some final view (the integrated schema or the user's desired query) to the lower level structures used in implementing that final view. However, the facilities for controlling and influencing rule execution are likely to be quite different for schema integration, a days-long process with an expert user. Obtaining a global architecture appears

to be formidable challenge that will require broad knowledge of many tools.

Acknowledgements: This work has been funded in part by the International Financial Research Services Center at MIT, National Science Foundation Grant #IRI902189, Xerox Advanced Information Technology, and ETH-Zurich.

References

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[KBH89] L. Kerschberg, R. Baum, and J. Hung. Kortex: an expert database system shell for a knowledge-based entity-relationship model. In *The Conference on the Entity-Relationship Approach*, Toronto, 1989.

[KN89] M. Kracker and E. Neuhold. Schema independent query formulation. In *The Conference on the Entity-Relationship Approach*, Toronto, 1989.

[McC84] J. McCarthy. Scientific information = data + meta-data. In *Database Management: Proceedings of the Workshop November 1-2, U.S. Navy Postgraduate School, Monterey, California*, Department of Statistics Technical Report, Stanford University, 1984.

[SM91] M. Siegel and S. Madnick. A metadata approach to resolving semantic conflicts. In *Proceeding of the 17th International Conference on Very Large Data Bases*, September 1991.