

Using Unity to Semi-Automatically Integrate Relational Schema*

Ramon Lawrence
Department of Computer Science
University of Iowa
ramon-lawrence@uiowa.edu

Ken Barker
Department of Computer Science
University of Calgary
barker@cpsc.ucalgary.ca

Abstract

Unity is an architecture for integrating relational databases that performs three processes: metadata capture, semantic integration, and query formulation and execution. The foundation of the architecture is a naming methodology that allows concepts to be integrated across systems. Semantic naming of schema constructs increases automation during integration and provides users with physical and logical access transparency during query formulation.

1. Introduction

Although numerous architectures provide database interoperability, automating the integration process is still a major challenge. Unity is an integration system based on the semantic naming of schema constructs. By standardizing how names are constructed, related concepts with different structural representations can be integrated based on related semantic names. Further, similar to the universal relation model [2], Unity builds a logical, structurally-neutral integrated view of database concepts which greatly simplifies querying. The goal of the Unity prototype is to determine if increased automation is achievable by enforcing a standardized naming convention across systems.

The Unity architecture [1] consists of autonomous databases queried using a global software layer. This layer is responsible for querying using ODBC and integrating results returned. Integration is achieved using three processes:

- **Capture Process:** A capture process is independently performed at each data source to extract database metadata into a XML document called an X-Spec.
- **Integration Process:** The integration process combines X-Specs into a structurally-neutral hierarchy of database concepts called an integrated context view.
- **Query Process:** The user formulates queries on the integrated view that are mapped by the query processor to SQL, and then query results are integrated.

*This research is partially sponsored by a NSERC Research Grant (RGP-0105566) and TRLabs.

The *capture process* is the semi-automatic extraction and semantic naming of database schema information to produce an X-Spec. Extraction of metadata such as table and field names, keys, and join information is the first step. The second step is to assign to each table and field a *semantic name* that denotes its meaning. Semantic names are systematically produced by combining terms from a pre-defined term dictionary.

Briefly, a semantic name consists of one or more context terms (contained in brackets (“[.]”) related by either IS-A (represented using a “;”) or HAS-A (represented using a “;”) relationships and an optional property name term. The property name term is only used when the semantic name is describing a field. A *context* is a semantic name which is a container for properties. Thus, tables have context semantic names.

For example, consider the dictionary terms required to represent an order database: *Order, Product, Id, Name, Customer, Amount, Price, etc.* These terms are pre-defined in a dictionary for integrators to use to build semantic names describing schema elements. Like an XML term set, the sole purpose of the dictionary is to prevent naming conflicts, and the dictionary may be expanded to incorporate new concepts.

Given an order database of the form: $O(oid, cname)$ and $OP(oid, pid, amt, pr)$, Unity extracts known metadata and assigns initial semantic names. This semantic description process can be completed independently for each database. The role of the integrator is to refine the semantic names to model concepts appropriately:

System Name	Semantic Name
O	[Order]
oid	[Order] Id
cname	[Order;Customer] Name
OP	[Order;Product]
oid	[Order] Id
pid	[Order;Product] Id
amt	[Order;Product] Amount
pr	[Order;Product] Price

The *integration process* combines one or more X-Specs by matching semantic names. Naming conflicts are prevented using the pre-defined terms. Some structural con-

licts are resolved when producing the structurally-neutral context view, while others are handled during query generation. Consider integrating a second order database $O_2(oid, FirstName, LastName, pid, amt, pr)$. Resolution of the composite versus single attribute conflict present in the modeling of a customer name is achieved by defining dictionary terms Last Name and First Name, promoting the single property Name to a context with properties, and creating appropriate translation functions.

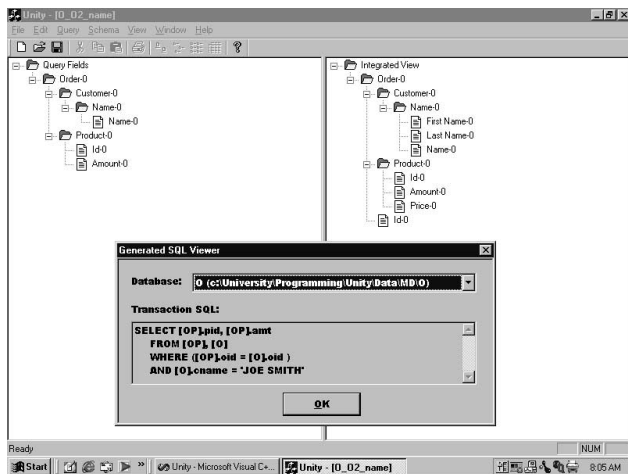


Figure 1. Querying View of O and O_2

Users query the context view by selecting semantic names. In Figure 1, the semantic query Q retrieves product id and quantities for customers named ‘Joe Smith’ and is mapped to SQL for each database as follows:

```
select [Order;Product] Id, [Order;Product] Amount
where [Order;Customer;Name] Name = ‘Joe Smith’ (Q)
```

```
select pid, amt from O, OP
where cname = ‘Joe Smith’ AND O.oid = OP.oid (QO)
```

```
select pid, amt from O2
where FirstName=‘Joe’ AND LastName = ‘Smith’ (QO2)
```

Mappings can be automatically generated at query time by utilizing semantic to system name mappings provided in X-Specs and translation functions available in the dictionary. The challenge in building the query processor involves handling ambiguous joins [3] as users often do not directly specify joins between contexts (tables).

2. Prototype Implementation

The current prototype is an all-in-one integration tool allowing a PC-based designer to integrate and query databases from one computer. The semantic capture component extracts metadata using ODBC and provides an environment for an integrator to build an X-Spec to describe

data source semantics. Although the prototype attempts to automatically build an X-Spec, designer input is typically required to specify key and join information that cannot be adequately extracted and assign appropriate semantic names to terms. The integration and query components allow the designer to manipulate the integrated view and generate test queries.

In this demonstration, we show that by accepting a pre-defined term dictionary, the construction of an integrated view can be increasingly automated. A major advantage with semantic naming is that querying the integrated view is less complex. Once the integrated view is constructed, Unity queries individual databases without the need for human designers to write wrappers, logic formalisms, or query programs. We have defined a standard term dictionary for our integration examples, but general integration can be realized by expanding the dictionary to contain terms relevant to the integration domain. Obviously, acceptance of a term dictionary involves numerous issues, but the growing use of XML with its own requirement of standardized naming shows that it may be practical in certain domains.

3. Conclusions

Unity implements a more pragmatic approach to schema integration. Although the integration and query facilities are not as powerful as mediator architectures because they lack explicit designer control, utilizing a standardized naming convention semi-automates integration and simplifies querying. By separating the specification of database semantics from the integration procedure, Unity implements automatic procedures to combine semantic specifications and resolve conflicts. The key benefit of the architecture is that the integration of data sources is automatic once the capture processes are completed. The capture process itself is partially automated by extracting metadata via ODBC and initial semantic name assignment.

Unity contains all the necessary architecture components for providing integrated relational database querying. The future goal is to extract the query and integration components into a Java-based implementation that can be used either as a web-browser plug-in or directly in Java programs using JDBC.

References

- [1] R. Lawrence and K. Barker. Integrating Relational Database Schemas using a Standardized Dictionary. In *SAC’2001-ACM Symposium on Applied Computing*, pages 225–230, Mar. 2001.
- [2] D. Maier, M. Vardi, and J. Ullman. On the Foundations of the Universal Relation Model. *ACM Transactions on Database systems*, 9(2):283–308, June 1984.
- [3] V. Owei and S. Navathe. Enriching the conceptual basis for query formulation through relationship semantics in databases. *Information Systems*, 26(6):445–475, 2001.