Mediator Languages – a Proposal for a Standard

Report of an I^3/POB working group held at the University of Maryland
April 12 and 13, 1996

Peter Buneman **Louiqa Raschid** **Jeffrey Ullman**

1 Introduction

The DARPA Intelligent Integration of Information (I^3) effort is based on the assumption that systems can easily exchange data. However, as a consequence of the rapid development of research, and prototype implementations, in this area, the initial outcome of this program appears to have been to produce a new set of systems. While they can perform certain advanced information integration tasks, they cannot easily communicate with each other.

With a view to understanding and solving this problem, there was a group discussion at the DARPA Intelligent Integration of Information/Persistent Object Bases (I^3/POB) meeting in San Diego, in January, 1996; and a further workshop was held on this topic at the University of Maryland in April, 1996. The list of participants is in Appendix A. The idea emerging from these meetings was *not* to force all systems to communicate according to specified standards, but to agree on the following:

- A *minimal core language*, or **Level 1** option, which would be a restriction of the object-oriented query language OQL, such that it will accept queries for relational databases. We recommend that all system components be able, at a minimum, to accept queries in this syntax, provided they address concepts (e.g., relations or classes, attributes or instance variables) known to that component. There must be a simple protocol to determine the schema of a system (its set of supported concepts).

- A simple format for representing answers. This could also be a fragment of OQL and will be included in the core language specification.

- A set of extensions, one of which could be full OQL, and would handle complex structures and abstract types (with methods). Other extensions will be needed to support rules (e.g., definitions of terms that can be shared among components), semistructured data (for self-describing objects), and shared code. A system component could support one or more of these extensions, independently, and there should be some simple protocol to determine the particular extensions that are supported.

1.1 The Problem Being Solved

One reason for wanting a core language emerges from the scenario of the *3–day crisis*, where a crisis management team has 3 days to build an I^3 system to integrate some predetermined set of independently developed information servers/systems, and then to incorporate new systems, that are identified as the crisis progresses. If every system has its own query language and its own representation for answers, there is a great amount of effort needed, by the system integrators, to understand the nuances of each system's language, and to write the appropriate translators.

In the proposed approach, much of the adaptation would be unnecessary. If a component were only needed for simple functions that could be described in the core language, then it would be only necessary for the integrators to learn the schema of the component. We do not want to underestimate the problems involved in understanding a schema, nor do we assure that a component accepts at least one of perhaps several standard meanings for its terms. The process of confirming that meanings are well understood will be present in any case, and subscription to standard ontologies has been advanced as the proper way to handle this problem.

Likewise, if advanced capabilities of a component were needed, the implementers could easily determine whether some component offered these capabilities. Exploiting these capabilities should be possible through one or more extensions of the core language, and there should be no need to learn a new core language just in order to use these capabilities.

1.2 Systems With Lesser Capability

The Maryland workshop also addressed the fact that some system components cannot usefully be described as answering queries in even a limited subset of SQL or OQL. An example would be an encryption system, whose sole purpose was to take a string and return an encrypted string. We therefore believe that it is essential to include, in the core, a **Level 0** option, in which a component is described as a function, giving only its input and output types, (string-to-string in the above example).

Even among more powerful components, we are concerned that, while they might appear to speak the entire core language over some set of concepts, in fact there are subtle limitations. For example, a bibliographic source might answer core-language queries over concepts title, author, publisher, etc., but it cannot answer the simple query `SELECT * FROM bibliography`, that asks for a complete copy of all its data. We are hopeful that there will be breakthroughs in developing some grammatical way to describe the actual family of queries that may be answered, or to systematically describe the constraints on the queries that may be asked.

At a minimum, such descriptions should be human-understandable, but it is interesting to speculate that the description could be made machine-understandable, to be used for example, by the query-optimizer/translator of another component. We observed at the workshop that OLE-DB, the "tabular" database interface extensions to OLE COM [Microsoft1994, Microsoft1996], appears to be heading in this direction.

2 Discussion

The Maryland workshop focussed on several issues and options. We did not attempt to resolve all issues, but we believe we have a framework for convergence.

2.1 Functionality

There was little disagreement that all data sources should be capable of expressing data in the relational model, and responding to a relational query language. We recognize that this may not hold for sources with limited capability, as discussed earlier. It was also agreed that the adopted core language should have the expressive power of the relational algebra, but no more.

It is important to note that query languages are based on a type system or data model. It follows that by requiring a data source to understand a certain language, we are also requiring it to map either all of its data, or at least all of the data needed to provide an answer to a query, into a data model appropriate to that language.

2.2 Syntax

For the purpose of implementing a core language, it is necessary to specify some syntax (for example nonrecursive Datalog with negation), but for this purpose alone, the issue of which syntax to use was not regarded as particularly important, since they are all equally easy to implement.

The motivation for discussing syntax further was whether, by adopting a specific syntax, we would make the core language more or less amenable to incorporating further functionalities. Understanding what these functionalities are, and why they are important, was one of the reasons for the Maryland meeting. The areas originally slated for discussion were the following:

- Exchange of rules
- Complex types and semi-structured data (self-describing objects)
- Abstract types (with methods)
- Exchange of Meta-data

In addition, early on in the Maryland meeting, it was decided that the following areas should also be discussed:

- Exchange format for the data.
- Communication protocol, including session control.

2.3 Alternative Approaches

Since the primary reason for wanting a core language is to facilitate the exchange of data, the immediate question is why not adopt one of the many existing standard data formats, e.g., ASN.1 [ISO1987], that have been developed for precisely this purpose. A simple *request* language would be needed to specify requests for data in this format. The problem with this solution is that most data sources do not simply dump all their data into some prescribed format; they usually respond to some request that describes *what* data is requested. Even if it were possible to obtain all this data, there may be problems of efficiency, and usability of this data. It is the language in which the requests are formulated – the query language – which was the focus of this discussion.

Another option that was rejected was to use KIF or another powerful language as the standard. While KIF may have its role representing rich knowledge sources, we believe it is too expressive to be a good model of most sources. What may be worse, components that are able to handle full first-order logical theories, or more, may behave unpredictably when given logical statements from a variety of sources with differing purposes.

3 Resolution

The main resolution at the Maryland meeting was to recommend the relational subset of OQL, the query language proposed by the ODMG committee, [Cattell96], as the core language. This sublanguage has a simple syntax – close to that of SQL – and implements the desired "core semantics" – that of the relational algebra. It also has the property that the full language supports a range of complex types (Set, Bag, List, Array), and allows free combination of these types. Thus, those sources that implement the full language will be able to communicate such data easily and efficiently. However, for the purposes of communicating with other agents that only speak the core sublanguage, it will be necessary to translate more complex types into relational format.

The OQL standard has definitions to support scalar and structured literals, and has type constructors; it can therefore describe relational data, and this is the input and output restriction of the core language. Hence, the proposed Level 1 core language provides a format for data transmission – another component of this proposal.

This proposal is based on several desiderata:

1. An existing standard should be used if possible. No-one wants another standards committee.
2. Interoperability with the core language should be something that most systems being developed within the I3 community (and others) can easily achieve.

In addition to these properties, OQL is a relatively simple language with compositional semantics. This means, for example, that wherever a relation name is used in an OQL program, a relation expression of the same type may be used. This property is an advantage in mediation, where it is desirable to have a language that can freely combine data from various sources. Thus, although the proposal is for a language for data exchange among mediators, the core language (with extensions), could also serve as a language for mediation.

The full report includes a BNF description of the core language and can be obtained at
`ftp://ftp.umiacs.umd.edu/pub/ONRrept/medmodel96.ps`

We now present brief summaries of the various working groups.

3.1 Complex types and semi-structured data

[Buneman, Ramakrishnan, Ullman]

It was the desire to have a language that would extend to "complex" data types that led to the adoption of OQL syntax. Complex types differ from relational types in two ways: first, relational databases only support sets and multisets (bags). Lists and arrays are two other widely used "collection types" that one would like to support in a database system. Second, relational databases only support "flat" data structures – sets or multisets of tuples of atomic types. Complex types allow these types to be freely combined; e.g. lists can be contained in tuples, which may in turn be members of sets.

Support for semi-structured data (self-describing data) is also a needed extension. At present, the minimal core language does support such descriptions (in a trivial manner), using the
<constructor_query> specification.

3.2 Abstract types (methods)

[Tannen, Qian, Raschid]

Abstract data types are necessary for any interchange data model since many simple sources essentially provide a functional interface. A typical example is a dictionary. Its interface is a function from strings (a word) to strings (its definition). Such an interface can be integrated with an OQL-like query language without impedance mismatch. Further, ODL-like class definitions can be thought of as abstract data types, where the methods are implemented using some particular language binding. In general, most object data models provide many additional features, most notably the support of OIDs and inheritance. However, class encapsulation of object data model and the ADT encapsulation is essentially similar. In the interchange data model for the mediator context, the implementation of the ADTs is the responsibility of the particular data sources (and their wrappers). The mediator data model only needs a linguistic mechanism for specifying ADT interfaces, i.e., a set of types, and a set of operations (for these types) must be specified, The types mentioned by the operations will include the ADTs, and also the standard types such as int, bool, string, real, etc. Thus, some convention about the representation of these standard types needs to be adopted, for example, a standard data format of [ISO1987]. The more interesting question is whether data of the types defined by the ADT can be exchanged? If so, is it feasible to have a common format for the exchange of ADT data? One possibility is to treat the operations of the ADT as constructors, as in the OQL query language. Then, the data can be treated as OQL expressions inolving these constructors.

3.3 Exchange of rules

[Abiteboul, Maier, Levy]

Two different motivations for the use of rules in mediators were investigated. The first is to use rules as a language to communicate various kinds of information between the mediator and a source. The second is to communicate knowledge, in the form of rules, between the sources and mediators. Extensions to the mediator model to support the exchange of rules is strongly influenced by the core language that is chosen, and, more importantly, by the role that rules play in such a language. We now enumerate examples of rules that exchange information and the exchange of information in the form of rules.

1. Constraints: Rules may express (integrity) constraints on the contents of a specific source; on the relationship between the contents of different sources; on the completeness of the contents of

the sources; etc. This information may be communicated to the mediator, using rules, and used for optimization, for example, to avoid sending irrelevant queries to a source.

2. Schema queries to a source: A mediator may require information about the *schema* of a particular source, so that queries appropriate to that schema may be sent. Rules can be used to query and exchange this schematic information.

3. Mapping and translation of *data* between a mediator and a source: This can include format translations and schema translations and restructuring. This information, in the form of specific rules, can be exchanged between a mediator and a source.

4. Integration: Rules may specify how information from multiple sources can be integrated with respect to the particular schema of the mediator. This information is also in the form of rules that are shared between the mediator and the sources.

5. Propagation of updates and monitoring: Rules may be used to specify how an update in a source should affect the answer to a mediator query, or to specify what changes (updates) in the sources must be monitored by the mediator.

6. Queries posed to the mediator or queries from mediator to sources may be expressed as a rule.

7. Rules support a compact format for representing information; instead of responding to a query with a set of all the answers, a source can respond with a rule, or intensional answer, that characterizes all the (extensional) answers.

3.4 Exchange of Meta-data and control issues

[Subrahmaniam, Zdonik]

The following two questions were explored with respect to meta-data:

1. What kind of meta-data characterizes a mediator system?

2. How should this meta-data be used in order to perform mediation tasks most effectively?

The following types of meta-data and their possible usages were identified

1. Structural Meta-data: Refers to the data structures (schema) of the data sources being integrated. This can be limited to data structures that are accessed external to the data source, or made public by a wrapper for the data source. Structural meta-data may be used to facilitate the task of software integration, and building parsers for handling data types returned by external processes.

2. Semantic Meta-data: Refers to information contained within the above mentioned data structures and specifies information about their semantic types, as well as inter-relationships between these types. Semantic meta-data may be used to infer that a given data item must be returned to the user in a form different from that in which it is stored. For example, the user asking for a total cost in US dollars may wish to convert multiple currency quotations into US dollars before aggregating the result. The existence of semantic meta-data would solve some problems of partially automating this process.

3. Cost Meta-data: Refers to information about the performance (either time or processing costs or financial charges) associated with different data sources. Cost meta-data is extremely useful in optimizing queries. For example, when evaluating a query plan, cost information about times/charges are invaluable in assessing the quality of the query plan.

4. Reliability Meta-data: Refers to information about the reliability of data sources; to how often the data contained in these data sources change with time; and if these data sources are maintained over time. As with cost meta-data, reliability meta-data is often useful in evaluating and rewriting a query plan. For instance, a (previous) answer can be stored and re-used, if it is known that the source is updated infrequently,

and if it is more expensive to recompute the answer. Similarly, a query plan may be rewritten to favor a reliable data source.

5. Ancestral Meta-data: Refers to generic information about the data source (who created, when, where and why?). May often be used to assess reliability of the source, or to assess the ability of a data source to provide certain kinds of information. For instance, one may choose to examine DoD databases for terrain information before turning to commercial GIS vendors, whose databases may be less attuned to a defense application.

3.5 Exchange protocols

It was suggested that we adopt **http** as the standard for exchange, in the manner of many online interfaces to IR search engines.

4 Further work

4.1 External Data Sources with Lesser Capability

There are inevitably some sources that cannot support the interface of the minimal core model, nor can they be "wrapped" to do this. Consider an on-line encyclopedia that associates words with meanings – its functionality is a mapping from input (character) strings to strings. In a relational database, this is represented as a binary relation on strings, however it may not be possible to extract the data in this form. Doing so would, for example, make it possible to copy the encyclopedia, as opposed to querying it.

In OQL and other systems, this limitation can be overcome by representing the encyclopedia as an external function. However, this solution may not easily extend to other sources, for example, sources that provide several indexing structures. This was one of the issues that requires further work.

More generally, the problem of describing what tasks a source can execute *efficiently* requires much further work. Data sources often implement several indices in order to support a variety of "canned" queries (though not a full query language). The problem of how to describe what a source can do efficiently, and of how to exploit this when building a wrapper, requires further work.

A proposed road-map was discussed, based on a progression from "canned queries", to functional descriptions or libraries, to templates or parameterized descriptions (views), to grammars, all of which are less powerful than the core Level 1 language.

References

[Cattell96] R.G.G. Cattell, et al. *The Object Database Standard - ODMG 93. Release 1.2.* Morgan Kaufmann, 1996.

[ISO1987] ISO, *Standard 8824. Information Processing Systems. Open Systems Interconnection. Specification of Abstraction Syntax Notation One (ASN.1)*, 1987.

[Microsoft1994] Microsoft, *OLE 2 Programmer's Reference*, Volumes 1-2, Microsoft Press, 1994.

[Microsoft1996] Microsoft, *The Component Object Model Specification*, Microsoft Development Library, 1996.

5 Appendix A List of Participants

Dr. Peter Buneman, University of Pennsylvania (co-chair)
Dr. Jeff Ullman, Stanford University (co-chair)
Dr. Louiqa Raschid, University of Maryland (Organizer)
Dr. Serge Abiteboul, Stanford University and INRIA, France
Dr. Alon Levy, AT&T Research
Dr. David Maier, Oregon Graduate Institute
Dr. Xiaolei Qian, Stanford Research Institute
Dr. Raghu Ramakrishnan, University of Wisconsin
Dr. V.S. Subrahmaniam, University of Maryland
Dr. Val Tannen, University of Pennsylvania
Dr. Stan Zdonik, Brown University