The Organization Engine: Virtual Data Integration

James S. Miller[1], Carl Niedner[1], and Jack London[2]
[1] Digital Equipment Corporation
[2] Fox Chase Cancer Center

ABSTRACT[1]

The Organization Engine is an early example of Virtual Data Integration — providing the appearance of integration at the desktop without modifying existing infrastructure. Starting with the Organization Engine, eight programming days were needed to provide uniform desktop access to a CODASYL-compliant hospital information system and to a MUMPS-based radiology information system (the technique is equally effective for relational and other data bases). The resulting tool provides a seamless integration of these two systems, image storage, pre-recorded audio, and document storage. In addition to providing uniform access, the tool allows healthcare providers to organize the data to suit their individual needs.

The ease of this integration lies in two simple techniques: the transformation of data from all sources into a single, homogeneous representation, and the use of simple customization files to describe new object types and formats. The approach is sufficiently general to allow the integration of applications which present external interfaces of radically different forms. Two such forms are discussed here: data map publication and transactions.

THE CHALLENGE OF INTEGRATION

Most healthcare institutions today use a profusion of disparate systems, each carefully designed for a single, specific purpose. Unfortunately, the majority of healthcare providers require uniform access to information dispersed among these various systems. Historically, several approaches to solving this problem have been tried: specific pair-wise solutions; centralized replication of departmental data; and interfacing standards. None of these approaches allows the solution to be tailored for (and possibly by) the individual. VIRTUAL DATA INTEGRATION is the apparent integration of information at the desktop, without modification to the existing infrastructure of niche applications, custom and standard interface protocols, and organizational boundaries.

All of the current approaches to solving the integration problem alleviate some of the problem, but they still suffer from several drawbacks:

1. None of them provides UNIFORM ACCESS to the data. The user must still be aware of the origin of the data, and must deal with data from different sources in different ways.
2. None of them provides CUSTOMIZED ORGANIZATION of the data. Each application program still specifies the way in which its data is accessed and viewed. It is not possible to intersperse data from one source with data from another.
3. None of them provides USER EXTENSIBILITY to accommodate user preferences or new data types.
4. They require extensive (and expensive) custom programming and continual program maintenance.

Any solution to the healthcare integration problem must address a specific set of requirements. First and foremost, it must solve these four problems. Second, it must capitalize on existing infrastructure — very few institutions can afford to discard their existing information technology investment. Third, it must be evolutionary and inexpensive. The Organization Engine (OE) is a prototype designed to demonstrate that virtual data integration satisfies all of these requirements.

The key to virtual data integration is providing each user with a single view of all data — regardless of such details as the data's location, storage format, metadata model, and so on — without disturbing the implementation of the various systems that manage the data. The choice of this single view is at the root of a successful virtual data integration effort. The OE explores one particular user view: all data is composed of a set of fields; each field has a value and an icon; the icons indicate a user-tailorable combination of the type and contents of the field. For example, the OE provides icons for "folders" of user-configured data, as well as "folders containing medical reports as filed in the hospital's medical computing system."

[1]This paper is a condensed version of CRL Technical Report 92/3, which can be obtained from:
D.E.C. Cambridge Research Lab
One Kendall Square, Building 700
Cambridge, MA 02139.

The current OE prototype is connected to two pre-existing healthcare applications as well as a variety of desktop utilities that deal with multimedia datatypes. Extending the set of datatypes is straightforward (see below). Most of this paper is devoted to the considerably harder task of integrating existing healthcare applications, which is demonstrated by examining the techniques used to integrate the DECrad radiology information system and the Fox Chase Cancer Center's medical computing service (MCS).

DECrad: DECrad provides hospital radiology departments with patient registration, exam tracking and scheduling, diagnostic report and film library management, accounting, and management reporting. DECrad is implemented in Digital Standard MUMPS (DSM), running on Digital's VAX/VMS platform[2].

MCS: The Medical Computer System developed at the Fox Chase Cancer Center (FCCC) provides standard hospital information system functionality, as well as capabilities that are of specific value in the treatment of cancer patients. The MCS provides the staff at FCCC with the ability to register patients; perform patient admissions, discharges, and transfers; transcribe radiology reports, histories, physicals, and discharge summaries; schedule diagnostic procedures and outpatient visits; and track the location of patient medical record charts and radiology films. Interfaces were written to commercial pharmacy and clinical laboratory applications. Data on services provided to patients are passed over a local area network to a commercial hospital billing package. Oncology-specific features of MCS include cancer staging, display of available treatment, drug side-effect information, and pre- and post-diagnostic workup guidelines. The MCS was built as a central clearinghouse and repository for all clinical information. It is a typical example of the central repository approach.

THE ORGANIZATION ENGINE

The Organization Engine began as a research project in Digital's Cambridge Research Laboratory. The goal was to provide a software base to explore the issues that arise from dealing with vast quantities of data, primarily located in "legacy repositories" — pre-existing systems that merge raw data with structuring information in individual, idiosyncratic ways. The software is divided into three distinct pieces, two of which were stable over a variety of information sources, and the third carefully tailored to each specific repository. The two generic pieces were a user interface and an integration toolkit which provides the connection between the user view of data and a transmission protocol. The repository-specific component implements this transmission protocol for a particular data repository. The design and implementation of the repository-specific component is described under "Server Interface Models."

User Interface

The OE specifies a modularity boundary between the user interface and the underlying integration system. This modularity allows modification of the user interface without affecting the integration system, and vice versa. The initial user interface is a "trial balloon" rather than integral to the design of the system.

The prototype's user interface encourages users to locate information by navigating rather than searching. The goal is to make it natural to put information in multiple places so that locating it later will be easy. "Copying" an object actually means copying a reference to the object, so it is very fast. In fact, the implementation allows some simple queries to be constructed and executed through this interface, but the user is unaware of these queries. From the user's perspective, the interface looks very much like a set of "blotters," each of which is similar to the Macintosh^{tm}FINDER or MS/DOS Windows^{tm}DESKTOP. The blotter contains various named icons corresponding to data objects, with the shape of the icon reflecting the operations that can be performed on the data and/or the origin of the data.

Using the mouse to double-click on an icon ACTIVATES the icon. The meaning of "activate" is quite flexible. A simple text file is used to customize the user interface; it maps the name of an icon to the operation that should be performed when it is activated. There is a standard operation that can be used to activate an icon to reveal another blotter (i.e. to make the icon behave like a folder). Another pair of operations allows an application (specified in the customization file) to be run; it is passed either the name or the contents of the icon as a command argument. This makes it simple to add data types: an icon is designed and the customization file is edited to map that icon to the application that handles the data type.

The Integration Toolkit

At the core of the OE is a toolkit used to connect the user interface with a wide variety of different data sources. See Figure 1. The user interface communicates with this toolkit through an API that allows the user interface to ask for the field names within a record; then for each of those it can ask for the contents and type of the field. It also allows the creation of new records, modification of fields, etc. The user interface specifies a record either by providing a record identifier received as the value of a field or a "magic" identifier specified by a data source as a root from which other records can be located.

User Interface			Records, fields
Organization Engine Tool Kit			UIDs Cookies
DECrad	FCCC MCS	Other	Stored data

Figure 1: Structure of the Organization Engine

The central toolkit is responsible for converting these operations into a smaller set that communicate back to the actual source of the data. There are only six operations at this end: initiate a connection with a data source, get a record (from the data source into the OE), create a new record, replace the contents of an entire record, replace the contents of a single field of a record, and close the connection to the data source. The toolkit communicates with the data source by specifying an identifier for records which previously came from that data source — just as the user interface communicates with the toolkit itself.

There is one important feature to notice about these interfaces: the side in possession of the data always specifies the identifier for the data. Thus, the user interface can only use identifiers that it received from the OE toolkit, and the OE toolkit can only use identifiers it received from the data source. The OE does *not* merely pass on these identifiers unchanged. Instead, it combines the identifier with a marker that allows it to identify the data source that produced the identifier. We refer to the identifiers passed between the OE and the data source as COOKIES, and the identifier passed between the Organization Engine and the user interface as OBJECT IDs. Thus, an object ID consists of two parts: a data source identifier and a cookie belonging to that data source. This technique resembles dynamic data typing[1], tagged records, and some object-oriented programming implementations[4].

The toolkit is intended as a body of code that must be extended to be useful. For each new data source, two things must be provided to integrate it into the OE framework: a service identifier (to be used by the OE to refer to data from the new source of information), and a small body of code implementing the six operations required by the OE. The next two sections provide some guidance in designing and building this code.

COMMUNICATIONS

The most important property of the communication mechanism is that it must be invisible to the user. There is probably no component of an overall system less interesting to the typical healthcare professional than the networks on which it is based. There is no set of problems less interesting to that professional than difficulties arising from interconnecting systems or networks. These mechanisms are very properly viewed as the portion of the system that the programming staff should hide.

The OE gives only a small amount of guidance here. First, the protocol used to connect to data sources is deliberately simple: six commands (open, get, create, reply, modify, close). The protocol makes no commitment to the location of the client/server split. The names of fields in individual records can be coded on either the client or server. The network transmission protocol is not specified by the toolkit, so this choice, too, can be made as part of the per-data-source integration code.

The current prototype has already adopted two different interconnection styles: one uses Remote Procedure Call to connect the toolkit to the data source, and the other uses DECnet task-to-task. Each of these mechanisms has advantages depending on the nature of the network and the data source. RPC is preferred when the network environment is multi-platform or multi-protocol, or if large amounts of data must be moved between the OE client and the data source server. DECnet task-to-task works well between two machines that both support DECnet and where the data to be moved is always in small amounts of predictable format.

SERVER INTERFACE MODELS

The hardest part of integrating a new data source using the OE toolkit is designing the overall structure of the server. This section describes two broad classes of server API architectures, the TRANSACTION-ORIENTED model and the DATA MAP PUBLICATION (DMP) model, and provides a comparison of the two.

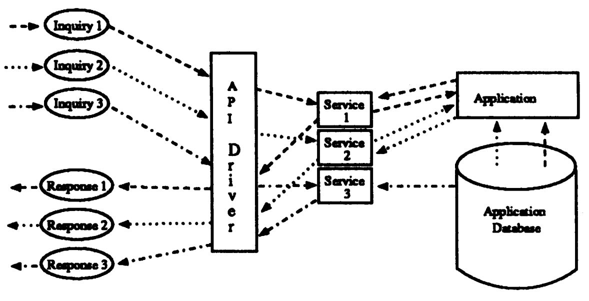

Figure 2: Transaction-Oriented Server

A transaction-oriented server allows information to be retrieved through a specific set of inquiry and reply transactions. Each inquiry message specifies the key values needed to retrieve the data; the type of the message determines the nature of the data retrieved. These servers are implemented by a top-level driver that determines the type of inquiry and dispatches routines to gather and format the information into the correct response. Such a server contains a separate code path for each inquiry message. Figure 2 shows the components of a transaction-oriented server.

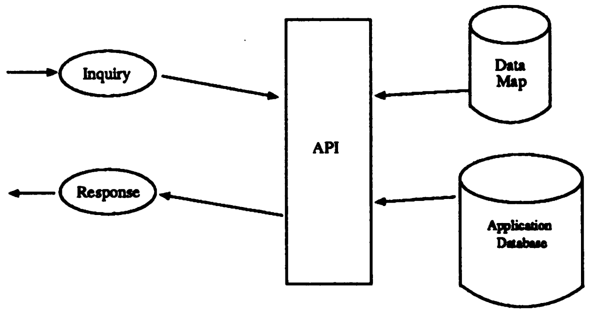

Figure 3: Data Map Publication Server

A DMP server makes use of a "map" of its application dataset which is distinct from the application itself. The data map represents a publishable subset of both the data elements in the application dataset and the relationships between those elements. The server uses this information to allow clients to browse at will among the published information. The top-level server code based on this model is more complex than the corresponding code for a transaction-oriented server. However, making the map explicit allows the server to be independent of the number and nature of data elements accessible through it. Figure 3 shows the components of a DMP server.

A data map is a form of metadata and contains elements in common with many different metadata representations. It may, in some cases, be derivable from existing metadata. A data map must support, at a minimum, the explicit representation of several relationships among data elements:

- **has-dependent** indicates that one data name (the object) is a DEPENDENT of another (the subject); i.e., the value of the object cannot be evaluated without knowing the value of the subject. The value of the object of a has-dependent attribute "becomes available for inspection" when the subject dataname's value is known.
- **reference:** for datanames with a single value, the object of this attributes specifies a retrieval method.
- **next-value-method:** for datanames with multiple values, the object of this attribute specifies a method for iterating through all of the values.
- **pointer-to** indicates this dataname is an "invisible link" between two other datanames. When the subject of this attribute is encountered as a dependent, its value should be assigned to the object dataname, and the object's dependents should be evaluated instead of the subject's.

It is convenient to represent the data map as a three level hierarchical index in which subject data names point to attributes, which point to object data names. This representation scheme is an instance of the entity-attribute-value data model[3]. In such a hierarchy, either all attributes for a given subject, or all objects for a specific attribute of a subject, can be retrieved quickly. For example, when an OE user selects an object, this scheme allows the DMP server to retrieve quickly all data names to be evaluated for display. The DMP server accomplishes this task by retrieving all objects of the HAS-DEPENDENT attribute for the data name corresponding to the selected object.

The choice between these two models is a trade-off: neither is clearly superior to the other. The decision must be made along three distinct dimensions, and evaluated for each data source to be integrated:

- **initial vs. continuing development cost.** The transaction model is easily built, and corresponds directly and naturally to many existing applications. Extending this model "across the network" to desktop computers is relatively easy, and this was precisely the approach taken by the Fox Chase Cancer Center in building their medical computing system. The DMP model requires a more complex initial system, since it requires code to implement and interpret the data map. Over time, however, the flexibility inherent in simply changing a data structure (the map) as opposed to writing new code (for new transactions) can become a dominant factor in development costs. In fact, the DECrad integration allows the data map to be changed while the interface is in use, and desktop users may see the effects of the change as soon as their next interaction.
- **control vs. flexibility.** If two different transactions consult the same underlying table, a transaction-oriented server can easily implement distinct access controls for the operations, while a DMP server would be almost powerless to enforce the distinction. On the other hand, the DMP model allows clients to browse the data in any order that they find useful.

- **performance.** A transaction-oriented server can achieve better throughput than a comparable DMP server, since the former dispatches retrieval and formatting code depending on the input message, while the latter must "interpret" the map. The performance difference depends largely on the complexity of the path that must be interpreted by the DMP server, and to a smaller degree on the complexity of the map itself.

OBSERVATIONS AND MEASUREMENTS

Experience to date has been far better than expected for a first attempt at combining a research prototype with pre-existing systems. The prototype integration project required a very small time expenditure: eight programming days to integrate two systems. The prototype is more than adequate for its intended purpose as a demonstration system. Initial indications are that the prototype may be useful in solving existing healthcare integration problems. With the low cost of integrating with other systems, the Organization Engine provides an interesting base for experimental user interfaces as well.

Quantitative performance characterization confirms that DMP servers can pay in performance for their superior flexibility. In an informal benchmark, the DECrad server retrieval logic, the only prototype component easily thus characterized, exhibited a nearly linear relationship between number of entities retrieved and elapsed CPU time. A MicroVAX II CPU (approximately 0.8 MIPS) required an average of 0.26 MIPS-seconds per retrieved entity (with a standard deviation of 0.036 MIPS-seconds). It is important to note that the quantitative results presented here were obtained without the benefit of any system tuning, and should not be used to predict operational characteristics of live systems.

CONCLUSIONS

A data map publication (DMP) server, which allows client applications to browse the public parts of a data set, provides a flexible base for virtual data integration. Building a DMP server is initially more difficult than building a transaction-oriented server, but the evolutionary path is simpler. It is possible to implement a DMP server for any application dataset that possesses the ability to:

- obtain a value from a dataset, given a singly-valued dataname and a list of prerequisite datanames and associated values
- iterate over the values for a multiply-valued dataname, with prerequisite information
- store the published data map, including dependency information, and invisible pointers.

Because a DMP server inherently requires the ability to interpret data map paths at runtime, it can be less efficient than a transaction-oriented server. It is not yet clear how serious this performance difference is in practice. Future work will both quantify this difference and reduce its current level.

Underlying this work is the toolkit provided by the Organization Engine. The structure of this toolkit has proven quite robust and is the main reason our modest labor investment in integration has been so effective. The Organization Engine provides a strong modularity boundary between the user interface and the underlying integration toolkit, allowing the replacement of the user interface without an overhaul of the entire system. The integration toolkit at the heart of the Organization Engine provides little more than the glue necessary to connect this user interface API with the simpler API used to connect with a data source. The user interface boundary deals with object identifiers, and provides an abstraction of records with fields. At the data source interface, the abstraction is one based on "cookies" and six fundamental operations: get, create, modify, replace, open connection, and close connection.

The virtual data integration method represents an effective, low-risk means for healthcare institutions to improve dramatically the accessibility of critical information without jeopardizing existing technology investments. The system described in the preceding sections is a prototype, not a finished product. Several issues remain to be addressed in future prototypes and pilot implementations, including security, multiplatform clients, inter-institution integration, client modification of server data, and others. However, the Organization Engine prototype presented here is exciting evidence that virtual data integration may help to solve some of the most critical information problems in healthcare today.

REFERENCES

[1] Harold Abelson and Gerald Jay Sussman, with Julie Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1985.

[2] Robert F. Davis. Radiology information system interfaces. In *Symp. on Comp. Appl. in Radiology.* Symposia Foundation Press, 1990.

[3] Carl D. Niedner. The entity-attribute-value model in radiology informatics. In *Symp. on Comp. Appl. in Radiology.* Symposia Foundation Press, 1990.

[4] Andrew Shalit. *Dylan: An object-oriented dynamic language.* Apple Computer, 1992.