Obtaining Information from Heterogeneous Systems

Gio Wiederhold

DARPA
on leave from
Stanford University
October 11, 1994

Between the minds that plan and the hands that build
there must be a mediator
Intertitle from Fritz Lang: *Metropolis*, UFA 1926.

Abstract
We visualize a three-layer architecture of private applications, mediating information servers, and information resources which may be databases, domain-specific programs and knowledgebases, sensor processing programs which report the current state, and simulations for projecting into the future.

A specific goal is that artificial intelligence programs remain simple and constrained to a single ontology. AI t asks in the mediating layer encompass focusing, pruning, fusion, and reduction. There are also control tasks associated with effective resource management. Results of mediation are to be composable by higher level applications, which also have to solve problems involving multiple heterogeneous subtasks.

The mediating modules will need a machine-friendly interface to support those applications. This interface must provide good communication, while encapsulating the artificial intelligence tasks, so that the complexity of the composed system is not much greater than that of the individual subtasks.

The corresponding architecture is a generalization of a server-client model. The partitioning enhances maintainability, but questions of effectiveness and efficiency do arise. The architecture is intended to support evolution by avoiding massive bindings through independent modularization.

This research was supported by DARPA N39-84-C211 the National Library of Medicine (LM04836), IBM and Hewlett-Packard Co's.

This note deals with resolving a critical conflict that arises when building large software systems. These systems have both the requirement to be adequately efficient in execution and sufficiently flexible to be maintainable in a changing world. Their scale is assumed to be such that multiple processors, multiple applications, and multiple types of software modules will be simultaneously active. Two critical points are elucidated

1 The high level application must obtain performance estimates as well as results from lower modules to create efficient schedules for each application type.
2 There will be no single optimization model or overall schedule, but this task will be as distributed and parallelized as the computations themselves.

These directions will be effective in the resource rich environment of the future, where the stress will be on application performance and flexibility to deal with changes.

1. Introduction

We define information systems as computer-based systems which can access a variety of computer-stored or generated base data, and select and process that data to provide specific information to aid planners and management in their decision-making.

We will hence assume a *horizontal* partitioning, where complex processing is allocated to a mediating layer. The mediators will draw upon resources from a voluminous base layer containing private and public resources. Applications will have to invoke multiple mediators, as sketched below.

		result → decision making	
Layer 3	Independent	**decision makers**	— managed on workstations
	- - - -	network services to information servers - - - -	
Layer 2	Multiple	**mediators**	— managed by domain specialists
	- - - -	network services to data servers - - - - -	
Layer 1	Multiple	**resources**	— managed in server systems
		input ← real-world changes and simulations	

Figure 1: The three layers of this architecture.

1.1 Resources

To clarify the tasks that are needed in the mediator layer we summarize the types of resources which are available. Many of the resources are likely to be massive. They must use representations that are effective for algorithmic approaches. Typical are databases, large bodies of text, and simulations of physical processes using spatial or four-dimensional (space and time) representations. The resources, however, are active engines, to be invoked by modules in other layers. The processing in the resource modules is rarely very complex , but has to deal with large volumes. Where efficiency is a concerns algorithmic processes will often be preferred. However, the results of this basic algorithmic processing are rarely directly suitable for high-level decision-making.

1.2 Mediators

We need for intelligent processes to acquire from the resources manageable amounts of selected, focused, but possibly complex results. We use the term *Mediation* to cover a wide variety of functions that enhance stored data prior to their use in an application. Mediation makes the interface to the workstation-based processes intelligent, by dealing with representation and abstraction problems that one must face when trying to use the wide variety of today's data and knowledge resources.

Examples of mediation are found in many current information systems and include

- Transformation and subsetting of databases using view definitions and object templates. These techniques reorganize base data into new configurations, appropriate to specific users and applications.
- Methods to access and merge data from multiple databases. These have to compensate for mismatch at the level of database systems, database structure, and the representation and meaning of the actual data values. A frequent case of mismatch is due to differing temporal representations, say monthly budgets and weekly production figures. These methods may induce uncertainty in the results because of mismatched sources.
- Computations that support abstraction and generalization over underlying data. These are needed to bring data at a low level of detail to a higher level. Typical operations involve statistical summarization and searching for exceptions. Abstractions are also needed to resolve mismatches.
- Much information is available in the form of text. Most text-processing is limited today to selection and presentation. More can be done. Most routine reports tend to have a degree of structure that makes some analysis feasible. Developing standards for text representation will help, regularizing textual information for further processing and mediation.

Specific citations can be found in [Wiederhold:92]., but most readers will be able to recognize these functions from their experience.

Mediators obviously also have an active role, their function cannot be achieved by having simple . interface specifications, to be used by applications in accessing base resources. They contain knowledge structures to drive the transformations [Wiederhold:91]. Mediators may also store intermediate results, but then must manage the consistency of such caches versus the base information.

1.3 End-user applications

We envisage end-user invoking applications at their workstations. Applications obtain information by dealing with abstractions supported by the mediators, and not by accessing base data directly. Different tasks will require a different set such mediators. A language will be needed to provide flexibility in the interaction between the end-users' workstation and the mediators. We discussed the partitioning of artificial intelligence paradigms into pragmatics (at the user-workstation layer) and the formal infrastructure (in the mediation layer) in [Wiederhold:90].

1.4 Vertical partitioning.

At each of those layers, will be a vertical partitioning. There will be multiple user applications, each using various configurations of mediators. Each of the mediators in turn will use distinct views over one or several base resources. The vertical partitioning creates a hierarchical binding only for one application instance, but not for the overall system structure. Just as databases today are justified by a diversity of shared usage, mediators should be sharable. Note that today's expert systems are rarely modular and sharable, so that their development and maintenance cost is harder to amortize.

For instance, the mediation module which can deal with inflation adjustment can be used by many applications. The mediator which understands postal codes and town names can be used by the post office, express delivery services, and corporate mail rooms.

We foresee here an incentive for a variety of specialists to develop powerful, but generally useful mediators, that can be used by multiple customers. Placing one's knowledge into a mediator can be more rapidly effective, and perhaps more rewarding, than writing a book on the topic.

1.5 Interfaces

We have assigned modules functionally to three layers. Only the application modules need to interact with end-user. Mediator modules need machine-friendly interfaces to support the applications, acquisition interfaces to obtain information from the base resources — which are often pre-existing, so that some of these interfaces have to adapt to legacy systems, and an expert's interface for the management of the knowledge they contain.

Databases commonly have interfaces that can be used by other computing modules, with some standards (as SQL), that were unfortunately specified before sufficient experience was gathered in composite systems. Simulations, sensor-based processing, and knowledge-based systems have less well established interfaces, although work in this area is in progress [Neches:91].

All these components are positioned along a *data highway*, provided by modern communication technology. The interactions of the components will be primarily constrained by logical and informational limitations, and not by physical linkages.

We have described a demonstration system in [Wiederhold:90]

The partitioned architecture is well suited to the modern multiprocessor networked environment.

2. The Base Conflict

While at first the two terms: modularity and integration, like small and large, seem to be at odds with each other, some reflection rapidly shows that in order to achieve integration, modularity is essential. Since integration of systems both increases the size and the number and diversity of objectives they must satisfy, modularity is the only way to manage and maintain such systems.

This note is not intended to dwell on this obvious point, but rather, given this point, to focus on the issue of what is needed to make such systems adequately flexible and efficient. Here we have another conflict. Flexibility implies late binding and convenient rearrangement and replacement of modules to serve changing objectives and changing environments. Efficiency implies early bindings and optimal arrangements of subtasks. It is achieved when methods which restrict the solution space are given priority and when parallelism is exploited to the best extent possible.

3. Interfaces

In large systems, we will have large modules, so that multiple computers will participate in any computation and parallelism becomes an intrinsic issue. The interface between software modules is also most effective if this interface is congruent to the hardware interface [Lee:91]. Such congruency permits us to move closer to the 'shared nothing' concept [Stonebraker:86], which has been taken on by many workers dealing with distributed data. The congruency of the hardware and software interface means that no data, other than arguments and results, have to be moved across the interface. Now these large modules need not to share data representations, other than again argument and result specifications. Their algorithms and inference engines can be specialized to the task at hand. Their relative independency also helps in keeping them maintained. Ability to carry out maintenance expeditiously will not only keep them current and useful, but also permits rework to keep them efficient.

4. Scheduling Modules for Optimization

We now come to the specific topic for this note. We are composing systems out of modules, and will hence want that the modules are efficient as well as powerful. The modularity of the approach is partially motivated by the ability to choose alternative modules or to create new ones, if changes in technology or functional requirements make that desirable.

However, efficiency of modules is not the only condition for having efficient systems. A system will be composed of many modules and the sequence and overlap of their execution is the major factor to achieve good performance. The concepts for arranging modules for optimal execution do not differ from those used in programming languages, databases [Dadaism], and artificial intelligence processing [Smith:86] However, such optimizers depend on models of the computations being performed. The models are evaluated with parameters being extracted or estimated, as the count of loop executions, the selectivity of a database predicate, the partitioning of the search space.

Optimizing the execution of modules which use a variety of computational techniques cannot be driven by a single model. This means that the optimizer must now also work at a higher level of abstraction. Now, instead of obtaining parameters for its model from the lower layers, this optimizer must import estimates of performance, that do not depend on understanding the type of computation below. Such estimates are best composed of expected values and their variance or uncertainty, so that the high level optimizer can also judge risk [WWC:91]. The expected value and variance is applied to two parameters: cost within the module and result size, leading to a quadruple. The essential point here is then: to build integrated systems with high-level functionality at the top layer requires that the support functions that enable efficient processing must also be moved to that higher level.

5. Alternative Architectures

The Starburst Project recognizes the need for its optimizer to be extendible to new methods [Widom:91]. It provides this extendibility by letting the implementor of new operations, also add rules to the optimizer pertaining to those methods. However, in a world where new methods are effectively remote - encapsulated in objects, located on remote nodes, placing the information for optimization at a high level central node is not appropriate. Furthermore, since distinct applications will use different initial nodes, one cannot even identify such a central node. Replication of optimization information is, of course, feasible, but that means that nodes will be loaded with optimization rules and models for component nodes they never use.

```
Tasks
                 *                   *              *
               *   *               *   *          *   *
             *   |   *           *   |   *       * |     *
           *     V     *       *     V     *     * V        *
         *   requests    *   *               * *               *
       *                   *                   *                 *
     *   estimates      results               * *                  *
   * +       +         *   + *    +    +    * + *+        +       +   *
 *   |       |       *     |    * |    |   *   |  |*      |       |
 *********************************         *******************************
                   *************************************
Resources
```

Figure 2. Application hierarchies and their interaction.

6. Multiple Schedulers

Another important point must be made explicitly now: there is not a single hierarchy in such an architecture. In every application a different set of modules can be assembled. Typically an application is partitioned top-down into a hierarchy that represents its specific objectives and the information and methods needed to support it. In terms of the optimization of processing that means that for each application very different schedules will be generated for the invocation and execution of these modules. While the optimizer code being used may be identical, the objectives, components, and results will differ. The extent of prebinding will differ as well; some application will have objectives will give more weight to flexibility, and others to performance. Performance can be enhanced, as always, by caching, by precomputing and storing results and abstractions, etc.

6.1 Interaction during optimization

The scheduler then can live at many nodes. It will not be affected by having new methods or entire computational nodes inserted into the set of system resources. The overhead for generating estimates will be low for large tasks. For small tasks, returning the results with the estimates can keep the overhead modest, while providing results already during the optimization phase, reducing subsequent costs. The optimization and execution phase boundary is now broken.

6.2 Distributed estimation

Overall this architecture avoids centralization. Individual applications are optimized for their performance. Lower level modules may serve multiple applications simultaneously. Their estimates, at execution time, may reflect reduced performance due to the committed load, and the application schedules may want to avoid overloaded modules, if possible. Computational modules, say simulations, can be replicated, if access to the input data is easy. Database modules cannot be replicated easily, so here it is important to separate computational and retrieval functions, so that their performance remains high. Multiple layers of services will appear in such an architecture, and estimates as well as data will be passed through the layers.

6.3 A critical shortcut

To make a realistic estimate, the input will, in most cases, be specified. This means that no new input supply interaction is needed to compute the result. Furthermore, the availability of input enables the shortcut alluded to above. When the cost of computation is low, the final result can be returned immediately The associated estimate, as well as its variance, is now precisely zero. The condition of having a complete input is especially likely for modules performing small tasks. No special distinction need be made for small versus costly execution modules. We have now a seamless scheme for small and large subtasks.

6.4 Blurring optimization and execution

Since some execution is initiated during optimization, the traditional boundary disappears. To what extent some or much optimization is best carried out as a precompilation phase, without known inputs, is unclear and awaits experience. If the cost of scheduling and composing large modules can remain relatively small then execution time optimization can be tolerated. On parallel machines, in fact, multiple candidate schedules can be started, and the less successful ones, when evident, aborted [Roy thesis].

7. The Computational Environment

This model of computation assumes a resource-rich environment, as envisaged in [HPCC:91]. There is no global optimization over all the modules, outside of the adaptation caused by applications rejecting use of overloaded modules. The optimization is oriented towards application performance, the critical issue in large-scale software. The estimation process envisaged can also serve real-time needs. The use of modules that create impossible schedules can be avoided if they can be replaced by modules producing less precise results faster.

8. The Research Task

The critical element missing today in such an architecture is the high-level language needed to compose applications from large scale modules, although we see some research results in application composition from large objects being relevant [ORB:91]. The composition task can be greatly aided by having tools as libraries with formal specifications, and graphical tools for the selection and insertion of modules [Hunt 90] .

To create a library of such modules we can initially wrap existing programs to serve within this architecture. The interface code to make them user-friendly can be omitted, greatly reducing their complexity. We foresee that the input-output code will move to the end-users workstations, which will then communicate with the root nodes of the application tasks [WWC:90], [Green:91].

We are envisaging today many large systems. Only if we have a partitionable and evolvable software architecture will we able to achieve such goals with the limited number and great diversity of individual contributors we have available.

References

[Green:91] M. Green and R. Jacob: "Software Architectures and Metaphors for Non-WIMP User-Interfaces"; Computer Graphics, Vol.25 No.3, Jul.1991, pp.119–235.

[HPCC:91] Committee on Physical, Mathematical, and Engineering Sciences "Grand Challenges: High Performance Computing and Communications" Supplement to President's FY 1992 Budget, 1991.

[Hunt 90] Neil Hunt "IDF, a graphical dataflow processing language for image processing, Proc. IEEE Conf. on Systems, Man and Cybernetics, Nov.1990.

[ORB:91] OMG "The Common Object Request Broker, Architecture and Specification" OMG Doc. 91.8.1, Aug.1991, DEC, HP, Hyperdesk Co., NCR, Object Design, Sunsoft.

[Lee:90] Byung Suk Lee "Efficiency in Instantiating Objects from Relational Databases through Views" PhD Thesis; Stanford Report CS-90-1346, Dec.1990.

[Neches:91] Robert Neches, Richard Fikes, Tim Finin, Tom Gruber, Ramesh Patil, Ted Senator, and William R. Swartout: "Enabling Technologies for Knowledge Sharing'; AI Magazine, Vol.12 no.3, Fall 1991.

[Stonebraker:86] Michael Stonebraker "The Case for Shared Nothing" IEEE Database Engineering, Vol.5, 1986, pp.4–9.

[Smith:86] David E. Smith "Controlling Inference" PhD Thesis, Stanford, CSD, CS-86-1107, Apr.1986.

[Wiederhold:90] G. Wiederhold, P. Rathmann, T. Barsalou, B-S. Lee, and D. Quass: "Partitioning and Combining Knowledge"; *Information Systems*, Vol.15 no.1, 1990, pp. 61–72.

[Wiederhold:91] Gio Wiederhold "The Roles of Artificial Intelligence in Information Systems"; Ras (ed): Lecture Notes in Artificial intelligence, Springer Verlag, 1991, pp. 38-51

[Wiederhold:92] Gio Wiederhold: "Mediators in the Architecture of Future Information Systems"; to appear in IEEE Computer, March 1992.

[Widom:91] J.Widom, R.J. Cochrane, and B.G. Lindsay "Implementing Set-Oriented Production Rules as an Extension to Starburst" VLDB 91, Barcelona, Sept.1991.

[WWC:90] Gio Wiederhold, Peter Wegner, and Stefano Ceri "Towards Megaprogramming" Stanford Report CS-90-1341 and Brown Un. report 90-20, Oct.1990.