RJ7515 (70030) June 7, 1990
Computer Science

IBM Research Report

The Melampus Project: Toward an Omniscient Computing Systems

Luis-Felipe Cabrera, Laura Haas, Joel Richardson,
Peter Schwarz, Jim Stamos
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

RJ 7515 (70030) 6/7/90
Computer Science

The Melampus Project: Toward An Omniscient Computing System

Luis-Felipe Cabrera
Laura Haas
Joel Richardson
Peter Schwarz
Jim Stamos

Computer Science Department
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract:

Software technology has not evolved to accommodate the rapidly increasing volume and diversity of on-line data. Finding data is often very difficult, and once found, data from different sources is often difficult to combine because of a lack of common formats and semantics. The advent of distributed systems has highlighted the problem by multiplying the amount of data that is hard to find and hard to use. A fundamental problem contributing to these shortcomings is the lack of a comprehensive model in which to describe and manipulate the range of entities in a system. This paper introduces Melampus, a system designed to address this problem.

The overall goal of Melampus is to define a powerful data model, to build a prototype system that implements the model, and to build sample applications that highlight both strengths and weaknesses of the model. We hypothesize that by providing a system-wide framework in which to describe the structure and behavior of data and by providing efficient associative access to all entities, we will significantly improve the ability of users to exploit the system. By providing global system coherence, we will enable the use of data in unanticipated ways, allow rapid retrieval of data, ease the formation of new relationships among data, and promote the sharing of data between applications.

CONTENTS

1. PREFACE

Melampus. A great seer in the Greek mythology who founded an important family of prophets; he could understand the speech of animals and birds; cured impotence and madness. [1]

Associative access. A value-based nonprocedural specification of a collection of entities.

Data. Uninterpreted *sets of bytes* present in a computer system. Meant to represent facts upon which an inference or an argument is based or from which an intellectual system of any sort is constructed. [2]

Information. (From the Latin word *informare*; meaning the action of forming matter.) An informing or being informed; formation or molding of the mind or character, training, instruction, teaching; communication of instructive knowledge. [3]

Knowledge. The fact or condition of knowing; the fact or condition of having information or of being learned or erudite; the fact or condition of knowing something with a considerable degree of familiarity gained through experience of or contact or association with the individual or thing so known. [2]

Omniscient. Knowing all things; possessed of universal or complete knowledge. [2]

User. A human using a computer.

Wisdom. The quality of having sound judgement, of being judicious, informed, learned, shrewd; ability to discern inner qualities and essential relationships; the intelligent application of learning. [2]

---

1 Gods and Mortals in Classical Mythology: A Dictionary. Michael Grant, Dorset Press, New York, 1985.

2 Webster's Third New International Dictionary, Unabridged.

3 Oxford English Dictionary.

2. MOTIVATION

The purpose of computing
is insight, not numbers.
Richard Hamming [26]

You say you want a revolution,
Well, you know
We all want to change the world.
John Lennon and Paul McCartney [42]

2.1. The Current World vs. The Ideal World

Computer systems today store and manipulate vast amounts of data. This data is isolated in many places: in the operating system kernel, in the network subsystem, in the file subsystem, in the database management system, and in applications. Unfortunately, such isolation often hinders the exploitation of data. This is a problem since the worth of such a resource is latent. The full value of data is realized only when it is used to respond to the needs and desires of a user, or when new information is derived by relating one piece of data to another. Isolation complicates finding a particular piece of data, since there are many places in which to look for it, and each subsystem provides a different way of asking for it. Isolation also makes interrelating data from different subsystems extremely difficult, since there is no agreement on representation or semantics.

The advent of networks and distributed systems has made the problem of exploiting data worse by increasing both the amount of data accessible and the number of places in which it can be stored. Moreover, each increase in hardware speeds (cpu, memory, and network) accentuates these difficulties by enabling new applications through the reduced cost of accessing and manipulating data. Thus, making effective use of the data stored in a computer system and using that data in unanticipated ways is already difficult. The prognosis is that the situation is going to get worse.

Envisioning the ideal world amounts to wishing away all of the perceived shortcomings of the current world. (The fact that perceptions change over time implies that even a perfect world, should it ever be implemented, would eventually be perceived as flawed. We will ignore this for now.) In the ideal computer system, data would not be isolated, nor would there be a multiplicity of ad hoc representations. Instead, a common data model would exist that would be flexible enough to allow the user to describe all data of interest, regardless of the application domain. Thus, everything, from operating system structures to program components to databases, would be described in the same model. This model would allow a user to find data easily, *i.e.*, it would provide a query language through which all entities in the system could potentially be accessed (modulo protection restrictions). Since such a system would be very large, there would be metadata that users could query to discover what entities were available in the system.

Flexible naming and access control schemes would allow users to tailor their environments, to share objects, and to define domains in which to pose queries.

Of course, the ideal system is both an ideal model and an ideal implementation. The system would be distributed, highly available, and would scale without bound. Optimization would be applied aggressively (and successfully!) throughout the system, yielding good performance. For example, the system might be adaptable, relocating and replicating objects in response to access patterns learned through usage. Finally, the system would be easy to install, use, and maintain, and it would serve as the day-to-day computing environment for a large user community.

2.2. The Melampus Project

We intend Melampus to be a step in the direction of the ideal world. We believe that the issue underlying the problems described earlier is fundamentally one of data modelling. Each new subsystem that defines its own way of representing and manipulating objects simply adds to the cacaphony of models present in the system. Furthermore, each subsystem is typically a closed world; the ad hoc data models provide no support for unanticipated queries, for relating objects across subsystems, or for system evolution. Melampus is an attempt to define, prototype, and experiment with a computing environment based on a better data model.

We initially plan to focus on defining three important aspects of the data model: the type system, the query language, and the naming scheme. A type system determines how objects may be structured and used; a query language allows sets of objects to be specified with a predicate instead of with explicit references; and a naming scheme determines how the world of objects may be organized. These three aspects are not orthogonal. For example, we intend to provide queries as first-class objects, *i.e.* to allow an object to be defined intentionally. Thus, the query language affects the type system in that the structure of an object may be defined by a query. Also, a query is defined in part by the objects over which it ranges. Thus, the naming scheme affects the query language. Section 2 discusses these and other issues in more detail.

We plan to build and experiment with the system once it has been designed. Our initial implementation will be a breadboard design that provides the functionality of the data model. We will also implement several application slices both to validate and to stress the data model. Although performance will be critical for a real system, we wish to fine-tune the model before we fine-tune its implementation. Prototyping the data model is an essential first step, since the best way to prove that the model leads to a significantly more usable system is to implement the system and to demonstrate its capabilities. However, while we have every intention of designing a powerful and elegant data model from the start, we also have no illusions about achieving perfection. Thus, building and using the system is also the only way to expose its shortcomings. Section 4 outlines our proposed time frame for project milestones.

2.3. Evolution vs. Revolution

Like many other projects, Melampus intends to provide a new computing environment. A fundamental design issue in such projects is how the new system will relate to existing systems. Proposed solutions generally fall into one of two categories that can be termed evolutionary and revolutionary [47]. An evolutionary path seeks to integrate existing software tools and databases. One way to implement this approach is to build a new subsystem that accesses the data and services of other subsystems; within the new "world," users view the external world through a common, higher-level model. The great advantage of an evolutionary path is that it preserves the existing world (a huge resource!) while providing a new, more unified way in which to view it. Since users are not required to abandon the familiar, they are typically more willing to embrace the new.

The revolutionary path discards (or discounts) existing systems and seeks to create a new, better world. The advantage to revolution is that one is liberated from the often conflicting constraints that come with trying to preserve existing systems and data. The researcher is thus free to rethink established concepts, to challenge accepted positions, and to avoid institutionalized mistakes. As a result, revolutionary projects often lead to important new concepts that capture the attention of even the most entrenched pragmatist. For example, object-oriented concepts have migrated from their origins in Simula [7] and Smalltalk [24] to C [33,26].

In the Melampus Project, we intend to take the revolutionary path to addressing the problems outlined at the beginning of this introduction. That is, the Melampus data model will define a new world that is not necessarily compatible with existing systems. We argue, however, that the distinction between evolution and revolution is really one of emphasis and is not a fundamental difference. Evolutionary paths, despite the preservation of existing systems, result from new thinking on the part of researchers and require a certain amount of new thinking on the part of users. Otherwise, they would be of little use. Revolutionary paths, despite the exclusion of existing systems, require bridges to the outside world. Otherwise, they would be of no use at all. The real difference, in our view, is that the evolutionary path concentrates on how to build the bridges while the revolutionary path concentrates on how to define the new world. In Melampus, we will be emphasizing first of all the definition of a new world. We will build bridges as necessary, but if we face a choice between raising the toll for crossing the bridge or compromising the design of the new world, we will elect the former.

3. RESEARCH APPROACH

To be conscious that you are ignorant of the facts
is a great step to knowledge.
Disraeli

3.1. Introduction

In this section, we detail the technical challenges that we plan to address during the initial phase of Melampus. These include, first and foremost, the definition of a type system that can describe the broad range of data present in a system and can support its evolution, a query language that allows efficient associative access to all data in Melampus, and a naming scheme that will allow users to organize their worlds and that is suitable for large scale distribution. To prove the power of Melampus, it is necessary to build applications that exploit the system, and this leads to another area for research. In this section, we describe these issues in more detail, state our goals, and present our plans. Our initial implementation will be a prototype that borrows as much existing support software (*e.g.*, an object repository) as we can find. We will omit the implementation (but not necessarily the design) of many pieces that would be necessary in a real system. At the end of this section, we briefly discuss how such topics might be investigated in later phases.

3.2. The data model

The central problems in Melampus are how to define a data model suitable for describing the structure and behavior of both real-world data and computer system data and how to provide efficient associative access to it. To this end, we are doing the following in parallel: reading and understanding the limitations of current research and commercial efforts, developing our own data model, and designing applications that will use this data model. We expect the applications to reveal possible limitations of the data model, thus helping us to refine it further.

3.2.1. The type system

One of the first problems to address is defining a suitable type system. The Melampus system must be able to incorporate new types of data, while at the same time promoting the sharing of existing data by multiple users and multiple applications in unanticipated ways. Object-oriented and relational database technologies will help us to achieve these goals. Object-oriented technology allows for an extensible set of abstractions that encapsulate both the representation and behavior of data. By means of a type system that separates implementation from specification, object-oriented technology encourages the definition of clean abstractions which can more easily be used in new contexts and integrated in new ways. The concepts of substitutability and inheritance should prove particularly helpful in this regard.

From the relational model of data we recognize the importance of having efficient associative access to all data and of having non-procedural query languages. In the relational model, however, value-matching forms the basis of all relationships, and hence, all relationships are implicit. There is no mechanism to represent complex interconnections among data explicitly. Object-oriented technology provides such a mechanism, namely, *object identity*. This facilitates information sharing, because a single piece of information can be used for several different purposes without replication and hence without inconsistency. Both explicit and implicit relationships are of great importance; we claim each is insufficient without the

other. The search for the Melampus data model will begin by considering an object-oriented approach. Adding better support for implicit relationships to the object-oriented approach is an interesting and important research area, and at least as promising as adding support for explicit relationships to the relational model [35].

The evolution of the structure and behavior of data is one of the most significant challenges faced by Melampus. One form of evolution modifies type definitions (*i.e.*, the schema). Such changes occur over time because of changing requirements or increased understanding. While there has been some work in this area [66,4,51], the problem is far from solved. Another form of evolution is the migration of individual objects through different types. Such migration is particularly important for those objects that model real world entities. For example, a person may be, at various times in his or her life, one or more of the following: student, employee, retiree, manager, or customer. Each of these roles can be considered a new type for the object, since each role carries with it a certain amount of characteristic state and behavior that persists as long as the role is valid for the person. Such natural evolution cannot be modelled in most object-oriented systems because a given object is constrained to have exactly one type throughout its lifetime. Type migration is only beginning to receive attention [23,63,68], and there are many open problems.

3.2.2. The query language

Our second task, intimately related with defining the type system, is to define a language in which to access and manipulate data. This language must be smoothly integrated into a Turing complete language, and it must support efficient associative access. Defining the Melampus language will require answering questions like: How do we specify the range of a query? How may we use queries as first-class objects to define objects intentionally? What language constructs should we include for exploiting the benefits of both navigational and non-procedural access to data? How should the language express explicit and implicit relationships among data? In particular, the query language should ease the access to any object in the system that satisfies access control constraints, regardless of type, intended use, or location. We will use our reading of the literature and the experience of department members to guide us in the language design and, again, we expect our application designs to help us refine the language constructs.

3.2.3. The naming scheme

As we described in the introduction, one of the problems with existing systems is the difficulty in finding data, and a fundamental issue in finding something is how to name it. A hierarchical name space organization has proved extremely useful both for file systems (*e.g.* Unix [58]) and network addressing (*e.g.* the Internet [57]). This organization allows related names to be grouped together as a unit, alleviates the problem of name clashes, and provides good scalability. Unfortunately, a strict hierarchy of names is often too constraining; one often would like the same object to belong to several different groups. Thus, for example, both hard and soft links are widely used in Unix. In addition, a naming hierarchy can actually prove cumbersome if the name of the desired object is not already known; one must traverse the hierarchy, looking in "likely" places.

In object-oriented database systems, a common way of organizing the name space is along the class lattice. That is, all of the objects of a given class are maintained in an *extent* for that class. The extent has the same name as its associated class. Since objects of all subclasses of some class T are, by definition, also T objects, the specification of an extent typically has two forms: one meaning all objects of exactly the class named, and the other meaning all objects of that class and all its subclasses. This kind of naming appears in O++ [2], Extra/Excess [16], Orion [34], Postgres [69], and O_2 [3].

Lattice-based naming provides a means for specifying the range of a query in an object-oriented system, but it is insufficient in a number of ways. First, it assigns names only to class extents; individual objects are not named. Both Extra/Excess and O_2 also support names for individual objects. Second, lattice-based naming does not support the tailoring of environments for individual users. The name space is essentially flat, with the entire lattice being the only naming context. Such an organization would be particularly cumbersome (not to mention difficult to maintain) in a widely distributed system.

3.3. The exploitation of the system

Complementary to the design of the data model is its exploitation in applications. We expect that implementing different applications in Melampus will expose unforeseen limitations that will pose new challenges. We will therefore implement slices of a variety of realistic applications and use what we learn to refine the model and its implementation.

The purpose of the applications is both to test and to showcase the data model. To test it, the applications should not be trivial. They should convince an objective observer that the Melampus data model can replace existing data models in current applications as well as support new applications for which the current data models are inadequate. Hence the applications must stress the traditional features of the data model (*e.g.*, object-orientation) as well as its novel features (*e.g.*, type migration). Although we do not expect to focus on efficiency in our initial implementation, the applications should eventually grow to stress the performance of the system.

To show off the potential of Melampus, the suite of applications must be sufficiently diverse that it would not currently be easy to draw relationships between the data they manage. Interrelating data from separate applications (enhancing an existing application or creating a new application) will help to demonstrate the power of the Melampus vision. Experimenting with the applications should also teach us how to exploit uniform associative access in order to create new and better applications.

There are also some pragmatic requirements associated with the Melampus application suite. Due to the limitation of existing resources, the design and implementation effort must not be too large. At least initially, therefore, the number and size of the application slices should be relatively small, and it must be easy to get the application data on-line.

It may not be possible to satisfy all of these requirements at once. However, some initial applications based on personnel databases, calendar management, and a small set of operating systems functions look promising. The personnel database and associated applications will demonstrate that Melampus can handle "traditional" database applications, and will show off novel aspects of the data model, such as type migration. Calendar management is a traditional office application, and calendar entries can be viewed as

complex objects (though with a relatively small amount of data), thus stressing the object-oriented capabilities of the model. Finally, the operating system functions will give us experience with data which varies more rapidly over time. This will stress the data model, and force us to think seriously about this type of performance issue (fast retrieval of complex objects is another performance issue we will have to address). As the Melampus vision demands uniform associative access to *all* data in the system, we need to develop techniques to deal with this type of data to make the vision a reality.

To make our application suite more concrete, consider the following collection of data and queries. The personnel database will include information on employees' titles, phone numbers, and offices. On top of this database we will build applications for presenting an organization chart and a telephone directory. The calendar management database will contain personal calendars, seating capacities for conference rooms, and calendars for conference rooms. We will provide simple applications for booking rooms and setting up meetings. From the operating system we will get information about the current set of users and processes.

Type migration can be used in the personnel database to model various roles such as employee, project leader, and manager. We can already think of several queries that exploit information from the different application areas. For example, the following query combines personnel information with operating system information: "Is anyone in my department running software package S at this very moment?" Another query combines information from all three areas: "Is there a convenient time and room for an afternoon meeting of the members of project P that have logged in today?"

The richness and diversity of these application databases will let us begin to explore the Melampus data model and vision. We will consider incorporating more databases and supporting larger applications in Melampus once we gain some experience with these simpler ones. Ideally, we will find others to cooperate with us on more ambitious applications that will really stress the model and the implementation and test the Melampus vision of integrating diverse applications. Possibilities for such applications include Rufus [48], and/or an advanced integrated programming environment.

3.4. Implementation

In the initial phase of the project, our implementation effort will stress functionality and fast turnaround over performance. We will prototype the system using whatever existing software platforms prove useful. For example, we will code new pieces in a rapid prototyping language such as Smalltalk-80 [24] or CLOS [9]. One particular subsystem that we will need is a persistent object store. Several research projects in the object-oriented database field have designed and implemented such general purpose stores. Examples include Mneme [49], ObServer (for Encore) [29], OM (for O_2) [71], and the Exodus Storage System [15]. One or more of these systems may be used in the initial prototyping of Melampus.

3.5. Future stages

The initial phase of the Melampus project will concentrate on defining the data model, while the second stage will involve building a prototype system, and experimenting with application slices. While

our experiences in these phases will be largely responsible for directing follow-on work, we expect that future phases will include some refinements to the data model. Hopefully, our iterations early in the design will keep such changes to a minimum. Thus, the bulk of the work in the next phase is likely to concern key system implementation issues. For example, query optimization in object-oriented databases is an open problem. Fast object materialization is another problem that is both interesting and important. Other implementation issues include distribution and high availability. Note that each of these areas could pose a significant research challenge in its own right. Therefore, an important step in charting the later phases of this project will be deciding whether we will attempt a research contribution in a given area or simply implement the best available solution.

3.6. Summary

The ultimate goal of the Melampus project is to provide a more coherent, usable computing environment. Key aspects of this vision are that the system provide associative access to all entities, that the system be able to model real-world entities, and that the system allow for wide scale distribution. The key areas of research for our initial phase include the definition of a flexible type system, query language, and naming scheme. Through prototyping the model and experimenting with several chosen applications, we will be able to refine and validate our design.

4. RELATED WORK

> Everything that can be invented has been invented.
> Charles H. Duell
> Commissioner U.S. Office of Patents, 1899

Melampus is not alone in trying to address the problem of global system incoherence. Many projects have sought to improve the world of computer systems, either through evolution or through revolution. Research efforts in database management systems, operating systems, persistent programming languages, and object-oriented languages have addressed, in isolation, many of the research topics implied by the ideal vision. In the design and implementation of Melampus, we will be able to draw upon many of the ideas (and hopefully, avoid many of the mistakes) of these systems.

4.1. Data Modelling

Any system that allows clients to create, manipulate, and destroy objects is essentially providing a model of data to the client. This model may or may not have a formal definition. In either case, the model defines how objects provided by the system may be structured and how they may be manipulated.

It is then up to the client to map the objects of its application domain onto the objects provided by the system.

Most operating systems, including the seminal Multics [22] and the influential Unix [58], assume that the structure of data is an uninterpreted byte stream. Moreover, they do not associate behavior with such unstructured data. It is thus very cumbersome, if not impossible, to organize all of the data in such a system. Plan 9 of AT&T has recognized this drawback [53,55,56]. The Plan 9 proposal is to add more semantics through use of additional file system naming. Thus, processes are file system objects, *i.e.* they are named subdirectories within the */proc* directory. Entries in such a process directory include file names for all the accessible components of a process, like its code and data segments and the **ioctl** signal mechanism. This approach, which extends the original Unix design to deal with an extensible set of objects, still falls short of providing the generality desired for large systems which have access to vast numbers of objects, as the human readable character string name is the only "query capability" provided to users and applications. There is no support for associative access to data, and there is still a limited amount of metadata.

Relational database management systems have traditionally provided such higher-level functionality. However, the relational model is widely perceived as too limited for new kinds of applications [35]. Thus, the design and implementation of object-oriented database systems (OODBs) is an active research area [1,2,3,16,23,45,69,73]. Some of the better-known projects include Encore [73], Iris [23], ODE [2], Orion [4], and O_2 [3]. There are also several commercial OODB products, such as GemStone [45] by Servio-Logic and VBase [1] by Ontologic. Our main complaint with these systems is that none provides the comprehensive data model that we are seeking. For example, with the exception of Iris, all provide fairly standard object-oriented models. Such models display shortcomings both in the modelling of real-world entities and in system implementation. (We enumerated some of these problems in the previous section.) Iris, while having an interesting data model, is computationally incomplete. Furthermore, none of these systems address the issue of organizing the world of objects in a large, distributed environment.

An important, open area of OODB research is the addition of associative, value-based access to objects, along with optimization techniques to make such accesses efficient. Iris [23], Extra/Excess [16], Encore [65], and Orion [35] have proposed query languages that bring some of the associative access capabilities of the relational model of data to the object-oriented domain. Other systems, such as ODE and O_2, have taken the approach of adding query facilities to an object-oriented programming language such as C++ [26].

Another important research area is in the evolution of systems that provide typed persistent objects. The central question is what happens to existing objects of a given type if the definition of that type is modified. This is a difficult problem, and only a few papers have appeared on the subject. Groups at Brown, MCC, and Servio Logic have proposed designs [66,4,51].

In addition to modelling the structure and behavior of entities, one other important aspect of the Melampus data model concerns distribution and naming, areas that traditionally have been addressed in operating system research. An important property of a distributed system concerns multi-site, atomic operations. Some file systems, such as Alpine [12] and QuickSilver DFS [14], have provided such multi-site atomic actions. Other systems like Auragen [10], Tandem's NonStop [5], and Isis [6] have addressed the more general problem of fault tolerant computing. Several languages have also been designed specifically to support the implementation of distributed systems. Examples include Argus [43] and Emerald [8].

Location independent naming is an important aspect of a distributed system since it separates naming an object from having to know its physical location. Not only are applications easier to write, but the system is free to relocate objects to improve performance. Several operating systems such as Amoeba [50], Domain [37], Locus [54], QuickSilver [14], Sprite [72], V [17], and and VAX Distributed Name Service [46], have implemented location independence.

Another aspect of naming in a distributed system is scaling. We wish the Melampus name space to remain manageable as the size of the system grows. File servers for Unix systems, most notably Andrew [30,61], and NFS from Sun Microsystems [44], have addressed this issue.

4.2. Other Related Work

The Rufus project [48] is, like Melampus, in the proposal stage, and it seeks to address similar problems. In particular, they wish to quell "information anxiety" through system-supported classification of text objects. The thrust of their research concerns ways to make explicit the implicit structure of a text (or byte-string) object so that the user can filter, organize, search, and edit such objects more flexibly. Rufus is an evolutionary project; it does not seek to remake the world, but to organize it. We view Rufus as complementary to Melampus. When we are ready to build elegant bridges between Melampus and the outside world, Rufus (or the technology they develop) may prove very useful.

Other related IBM software engineering efforts include the RPDE^3 project [27], the SAA/AD Repository Platform [32], and the VM/Software Engineering product [31]. The SAA/AD Repository Platform is a tool support layer that provides object-oriented services for the IBM Repository Manager. The platform represents and stores data using an entity/relationship paradigm on top of a relational storage manager. For performance reasons, the granularity of objects will be large. The VM/Software Engineering product is a central repository of source code, object code, and documents. It supports version and release control, answers database queries, and manages dependencies and relationships among objects. VM/Software Engineering stores some information in a file system and the rest in a relational database.

Another relevant research area concerns the transfer of database techniques for transactional concurrency control and recovery into the operating system. Camelot [67] and QuickSilver [28] have pursued this approach. These systems provide database-like services to a wide range of clients, although they are not concerned with providing sophisticated data modelling capabilities. Several other operating systems are either built on database management systems or provide the illusion that they are. For example the AS/400 [18,62] and PICK [11] systems fall into the former category, while the OONIX [13] and ROSI [36] systems fall into the latter category. None of these systems provide a means of extending the initial, system-defined set of data types with new, user-defined types.

There are a number of alternatives for implementing persistence in a system such as Melampus. Memory-resident database management technology may be a promising approach. Memory-resident data structures are made persistent by the recovery mechanism. Logging the changes to memory-resident objects can be performed using commit groups [19] or using the Stable Log Buffer approach [40]. Although work has been done in index structures [39], query processing [20,38,64,70], concurrency control [41,59], and recovery [21,25,40,60], these efforts are designed to support relational systems. These techniques may or may not be applicable in an object-oriented system.

5. RESEARCH PLANS

You can always tell the pioneers
by the arrows in their backs.
Overheard at Xerox PARC

To fail well, go out and
do something stupid.
Tom Peters [52]

We will start with a core group of available people, design and implement as a team, and eventually expand from both inside and outside IBM.

5.1. Melampus system

The following milestones for Melampus have been motivated and proposed in this document:

1. Definition of a data model. This includes as major subproblems: definition of a type system, definition of a query language, and definition of a naming scheme.
2. Initial design of chosen applications.
3. Evaluation of prototyping platforms, including data repositories, programming languages, and programming environments.
4. Prototype implementation of the data model.
5. Prototype implementation of chosen applications.
6. Evaluation of the system.
7. Extension and refinement of the system in directions determined by the evaluation.

A time frame for these milestones is the following:

- Phase 1. Completion by YE90 (8 months)
  1. Data model and query language defined.
  2. Slices of applications designed.
- Phase 2. Completion by 2Q92 (18 months)
  1. Prototype system implemented.
  2. Application slices implemented on prototype.
  3. Initial evaluation of data model.
- Phase 3. Possible future directions. (to be determined)
  1. Redesign data model based on experience in Phase 2.
  2. Design bridges to existing world.
  3. Throw prototype away; build a real system.
  4. Declare victory; move on to something else.

Acknowledgments. We would like to thank the many people who commented on earlier drafts of this paper. Their feedback helped us to clarify our goals and focus this proposal. Thanks to: Mike Carey, Dean Daniels, Roger Haskin, Bruce Lindsay, Eli Messinger, John Palmer, Dragutin Petrovic, Wayne Sawdon, Kurt Shoens, Irv Traiger, Shin-Yuan Tzou, Moshe Vardi, and Robin Williams.

6. REFERENCES

1. Andrews, Timothy, and Harris, Craig, Combining Language and Database Advances in an Object-Oriented Development Environment, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1987,* (Orlando, Florida, October 1987) pp. 430—440.

2. Agrawal, R., and Gehani, N., ODE: The Language and the Data Model, in *Proceedings of ACM SIGMOD 1989,* (Portland, Oregon, 1989) pp. 36—46.

3. Bancilhon, Francois, Barbedette, Giles, Benzaken, Veronique, Delobel, Claude, Gamerman, Sophie, Lecluse, Christophe, Pfeffer, Patrick, Richard, Philippe, and Velez, Fernando, The Design and Implementation of O_2, an Object-Oriented Database System, in *Proceedings of the Second International Workshop on OODBS,* Springer-Verlag Lecture Notes in Computer Science 334, (Badmunster, RFA, 1988) pp. 1—22.

4. Banerjee, J., Kim, W., Kim, H. J., and Korth, H., Semantics and Implementation of Schema Evolution in Object-Oriented Databases, in *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, May 1987) pp. 311—322.

5. Bartlett, J., A NonStop kernel, in *ACM Proceedings of the Eighth Symposium on Operating Systems Principles,* (Pacific Grove, California, December 1981) pp. 22—30.

6. Birman, Kenneth P., Replication and fault-tolerance in the ISIS system, in *Proceedings of the Tenth ACM Symposium on Operating Systems Principles,* (Orcas Island, Washington, December 1985) pp. 79—86.

7. Birtwistle, G., Dahl, O., Myhrtag, B., and Nygaard, K., Simula Begin, Auerbach Press (Philadelphia, 1973).

8. Black, A., Hutchinson, N., Jul, E., and Levy, H., Object Structure in the Emerald System, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1986,* (Portland, Oregon, September 1986).

9. Bobrow, D.G., DeMichiel, L.G., Gabriel, R.P., Keene, S.E., Kiczales, G., and Moon, D.A., Common Lisp Object System Specification X3J13 Document 88-002R, in *SIGPLAN Notices,* Special Issue, (September 1988).

10. Borg, Anita, Baumbach, Jim, and Glazer, Sam, A Message System Supporting Fault Tolerance, in *Proceedings of the Ninth ACM Symposium on Operating Systems Principles,* **17, 5** (November 1983) pp. 90—99.

11. Bourdon, Roger J., The PICK Operating System, A Practical Guide, Addison-Wesley Publishing Company (1988) pp. 1—450.

12. Brown, M. R., Kolling, K. N., and Taft, E. A., The Alpine file system, *ACM Transactions on Computer Systems,* **3, 4** (November 1985) pp. 261—293.

13. Bueche, Edward C., Franklin, Maurice T., Holley, Edward R., Korth, Henry F., and Sheppard, Gene C., Oonix: An Object-Oriented Unix Shell, in *Proceedings of the 22nd Hawaii International Conference on System Science, Volume II, Software Track,* (January 1989) pp. 928—935.

14. Cabrera, Luis-Felipe, and Wyllie, Jim, QuickSilver Distributed File Services: An Architecture for Horizontal Growth, in *Proceedings of the 2nd IEEE Conference on Computer Workstations,* (Santa Clara, California, March 1988) pp. 23—37.

15. Carey, Michael, Dewitt, David D., Richardson, Joel, and Shebita, Eugene., Object and File Management in the EXODUS Extensible DBMS, *Proceedings of VLDB 1986,* (Kyoto, Japan, August 1986) pp. 91—100.

16. Carey, M., DeWitt, David D., and Vandenberg, S., A Data Model and Query Language for EXODUS, in *Proceedings of ACM SIGMOD 1988,* (Chicago, Illinois, June 1988) pp. 413—423.

17. Cheriton, David R., The V Kernel: A Software Base for Distributed Systems, *IEEE Software,* **1, 2** (1984) pp. 19—43.

18. Clarck, B. E., and Corrigan, M. J., Application System/400 Performance Characteristics, in *IBM Systems Journal,* **28, 3** (1989) pp. 407—423.

19. DeWitt, David D., Katz, Randy H., Olken, Frank, Shapiro, L., Stonebraker, Michael, Wood, David, Implementation Techniques for Main Memory Database Systems, in *Proceedings of ACM SIGMOD 1984,* (June 1984) pp. 1—8.

20. DeWitt, David, and Gerber, R., Multiprocessor Hash-Based Join Algorithms, in *Proceedings of high 1985,* (August 1985) pp. 35—41.

21. Eich, M., A Classification and Comparison of Main Memory Database Recovery Techniques, in *Proceedings of the 1987 Database Engineering Conference,* (1987) pp. 332—339.

22. Feiertag, R. J., and Organick, E. I., The Multics Input-Output System, in *Proceedings of the Third ACM Symposium on Operating Systems Principles,* (1971) pp. 35—41.

23. Fishman, D. H., Beech, D., Cate, H. P., Chow, E. C., Connors, T., Davis, J. W., Derrett, N., Hoch, C. G., Kent, W., Lyngbaek, P., Mahbod, B., Neimat, M. A., Ryan, T. A., and Shan, M. C., Iris: an Object-Oriented Database Management System, in *ACM Transactions on Office Information Systems,* **5, 1** (January 1987) pp. 48—69.

24. Goldberg, Adele, and Robson, David, Smalltalk-80: The Language and Its Implementation, in *Addison-Wesley,* (Reading, Massachusetts, 1983).

25. Hagmann, Robert, A Crash Recovery Scheme for a Memory-Resident Database System, in *IEEE Transactions on Computers,* (1986).

26. Hamming, Richard W., Numerical Methods for Scientist and Engineers, McGraw-Hill (New York, 1962).

27. Harrison, William, RPDE^3: A Framework for Integrating Tool Fragments, in *IEEE Software,* (November 1987) pp. 46—56.

28. Haskin, R., Malachi, Y., Sawdon, W., and Chan, G., Recovery Management in QuickSilver, *ACM Transactions on Computer Systems,* **6, 1** (February 1988) pp. 82—108.

29. Hornick, M., and Zdonik, S., A Shared, Segmented Memory System for an Object-Oriented Database, *ACM Transactions on Office Information Systems,* **5, 1** (January 1987) pp. 70—95.

30. Howard, John H., Kazar, Michael L., Menees, Sherri G., Nichols, David A., Satyanarayanan, M., Sidebotham, Robert N., and West, Michael J., Scale and Performance in a Distributed File System, *ACM Transactions on Computer Systems,* **6, 1** (February 1988) pp. 51—81.

31. IBM Corporation, VM/Software Engineering General Information Manual, in *IBM Form No. GH21-0012-1,* (August 1987).

32. IBM Corporation, Repository Manager/MVS, in *IBM Form No. GC26-4608-8,* (September 1989).

33. Kernighan, B., and Ritchie, D., The C Programming Language, Prentice-Hall (1978).

34. Kim, Won, A Model of Queries for Object-Oriented Databases, in *Proceedings of VLDB Conference,* (Amsterdam/., August 1989).

35. Kim, Won, A New Database For New Times, *Datamation,* (January 15, 1990) pp. 35—42.

36. Korth, Henry F., and Silberschatz, Abraham, A User-Friendly Operating System Interface Based on the Relational Data Model, in *Proceedings of the International Symposium on New Directions in Computing,* (Augusr 1985) pp. 302—310.

37. Leach P. J., Levine, P. H., Douros, B. P., Hamilton, J. A., Nelson, D. L., and Stumpf, B. L., The architecture of an integrated local network, *IEEE Journal on Selected Areas in Communications,* **SAC-1, 5** (November 1983) pp. 842—857.

38. Lehman, Tobin J., and Carey, Michael, Query Processing in Memory-Resident Database Systems, in *Proceedings of ACM SIGMOD 1986,* (June 1986) pp. 239—250.

39. Lehman, Tobin J., and Carey, Michael, A Study of Index Structures for Memory-Resident Database Systems, in *Proceedings of VLDB 1986,* (Kyoto, Japan, August 1986) pp. 294—303.

40. Lehman, Tobin J., and Carey, Michael, A Recovery Algorithm for a High-Performance Memory-Resident Database System, in *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, June 1987) pp. 35–41.

41. Lehman, Tobin J., and Carey, Michael, A Concurrency Control Mechanism for Memory-Resident Database Systems, in *Foundations of Data Organization and Algorithms,* (June 1989) pp. 490–504.

42. Lennon, John, and McCartney, Paul, Revolution, on *The White Album,* Capitol Records.

43. Liskov, Barbara, and Scheifler, Robert, Guardians and Actions: Linguistic Support for Robust, Distributed Programs, in *ACM Transactions on Programming Languages and Systems,* (July 1983) pp. 381–404.

44. Lyon, B., and Sager, G., Overview of the Sun Network File System, Sun Microsystems, Inc., (Mountain View, California 94110, January 1985) pp. 1–8.

45. Maier, D., Stein, J., Otis, A., and Purdy, A., Development of an Object-Oriented DBMS, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1986,* (Portland, Oregon, September 1986).

46. Martin, Sally J., McCann, Janet M., and Oran, David R., Development of the VAX Distributed Name Service, *Digital Technical Journal,* **3, 9** (June 1989) pp. 9–15.

47. Messinger, Eli B., personal communication, 1989,.

48. Messinger, Eli, and Shoens, Kurt, The Rufus System: Organizing Information to Reduce Information Anxiety, Draft Report, (IBM Almaden Research Center, March 1990).

49. Moss, J. E. B., and Sinofsky, S., Managing Persistent Data with Mneme: Issues and Application of a Reliable, Shared Object Interface, *COINS Technical Report 88-30,* (University of Massachusetts, April 1988).

50. Mullender, Sape J., Editor, The Amoeba Distributed Operating System: Selected Papers 1984-1987, in *CWI Tract,* (Amsterdam, The Netherlands, 1987) pp. 1–309.

51. Penny, D.Jason, and Stein, Jacob, Class Modification in the GemStone Object-Oriented DBMS, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1987,* (Orlando, Florida, October 1987) pp. 111–117.

52. Peters, Tom, On Excellence, in *San Jose Mercury News,* (February 12, 1990).

53. Pike, Rob, and Thompson, Ken, Position Paper for IEEE Workshop on Operating Systems, in *Proceedings of First IEEE Workshop on Workstation Operating Systems,* (Cambridge, Massachussets, November 1987).

54. Popek, G., Walker, B., Chow, J., Edwards, D., Kline, C., Rudisin, G., and Thiel, G., LOCUS: A network transparent, high reliability distributed system, in *Proceedings of the Eighth ACM Symposium on Operating System Principles,* (Asilomar, California, December 1981) pp. 160–168.

55. Presotto, David, Plan 9, *Computer Science Seminar,* (IBM Almaden Research Center, 1988).

56. Presotto, David, Networking for Plan 9 from Bell Laboratories, *Spring 1988 EUUG Conference,* (London, England, April 1988).

57. Quarterman, John S., and Hoskins, Josiah C., Notable Computer Networks, *Communication of the ACM,* **29, 10** (October 1986) pp. 932—971.

58. Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *Communication of the ACM,* **19, 7** (July 1974) pp. 365—375.

59. Salem, K., Garcia-Molina, Hector, and Alonso, Rafael, Altruistic Locking: A Strategy for Coping with Long Lived Transactions, in *Proceedings of High Performance Transaction Systems,* (September 1987).

60. Salem, K., and Garcia-Molina, Hector, Crash Recovery Mechanisms for Main Storage Database Systems, Princeton University Computer Science Department, Technical Report 034086, (April 1986) pp. 35—41.

61. Satyanarayanan, M., Howard, J., Nichols, D., Sidebotham, R., Spector, A., and West, M., The ITC distributed file system: Principles and design, in *Proceedings of the Tenth ACM Symposium on Operating System Principles,* (Orcas Island, Washington, December 1985) pp. 35—50.

62. Scheicher D. L., and Taylor, R. L., System Overview of the Application System/400, in *IBM Systems Journal,* **28, 3** (1989) pp. 360—375.

63. Sciore, Edward, Object Specialization, in *ACM Transactions on Office Information Systems,* (April 1989) pp. 103—122.

64. Shapiro, L., Join Processing in Database Systems with Large Main Memories, in *ACM Transactions on Database Management Systems,* (September 1986) pp. 239—264.

65. Shaw, G., and Zdonik, S., A Query Algebra for Object-Oriented Databases, *Technical Report TR CS89-19,* (Department of Computer Science, Brown University, 1989).

66. Skarra, A., and Zdonik, S., The Management of Changing Types in an Object-Oriented Databases, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1986,* (September 1986) pp. 483—495.

67. Spector, Alfred Z., Thompson, Dean, Pausch, Randy F., Eppinger, Jeffrey L., Duchamp, Dan, Draves, Richard, Daniels, Dean S., and Bloch, Joshua J., Camelot: A Distributed Transaction Facility for Mach and the Internet---An Interim Report, *Carnegie Mellon Technical Report CMU-CS-87-129,* Department of Computer Science, Carnegie Mellon University, (Pittsburgh, PA, June 1987).

68. Stein, Lynn Andrea, and Zdonik, Stan, Clovers: The Dynamic Behavior of Types and Instances, *Technical Report TR CS-89-42,* (Department of Computer Science, Brown University, 1989).

69. Stonebraker, Michael, and Rowe, Larry, The Design of POSTGRES, in *Proceedings of ACM SIGMOD 1986,* (Washington, D.C., May 1986) pp. 340—355.

70. Swami, A., A Validated Cost Model for Main Memory Databases, in *Proceedings of ACM SIGMETRICS 1989,* (May 1989).

71. Velez, F., Bernard, G., and Darnis, V., The O_2 Object Manager: an Overview, *ALTAIR Technical Report,* (1989).

72. Welch, Brent B., and Ousterhout, John, Prefix Tables: A simple mechanism for locating files in a distributed system, in *Proceedings of the Sixth IEEE International Conference on Distributed Computing Systems,* (Cambridge, Massachusetts, May 1986) pp. 184—189.

73. Zdonik, S., and Wegner, P., Language and Methodology for Object-Oriented Database Environments, in *Proceedings of the Hawaii International Conference on System Science,* (January 1986).

The following references are also related to Melampus, although they were not directly cited in the text.

7. ADDITIONAL REFERENCES

1. Atkinson, M. P., Bailey, P. J., Chicholm, K. J., Cockshott, P. W., and Morrison, Ron, An Approach to Persistent Programming, in *The Computer Journal,* **26, 4** (1983) pp. 360—365.

2. Balzer, Robert M., Living in the Next-Generation Operating System, *IEEE Software,* The article is reprinted from Information Processing 86, H. J. Kugler, ed., North-Holland, Amsterdam, 1986. (November 1987) pp. 77—85.

3. Balzer, Robert M., Models of Database Systems, in *Proceedings of the 1989 SIGMOD Workshop on Software CAD Databases,* (Napa Valley, California, February 1989) pp. 5—8.

4. Batory, D., GENESIS: A Project to Develop an Extensible Database Management System, in *Proceedings 1986 International Workshop on Object-oriented Database Systems,* (Asilomar, California, September 1986).

5. Carey, M., DeWitt, D., Graefe, G., Haight, D., Richardson, J., Schuh, D., Shekita, E., and Vandenberg, S., The EXODUS Extensible DBMS Project: An Overview, in *Readings in Object-Oriented Databases,* S. Zdonik and D. Maier, Editors, (Morgan-Kaufman, 1989).

6. Haas, L., Cody, W., Freytag, J., Lapis, G., Lindsay, B., Lohman, G., and Pirahesh, H., An Extensible Processor for an Extended Relational Query Language, *IBM Research Report RJ 6182,* (April 1988).

7. Haas, L., Chang, W., Lohman, G. M., McPherson, J., Wilms, P. F., Lapis, G., Pirahesh, H., Lindsay, B., Carey, M., and Shekita, E., Starburst Mid-Flight: As the Dust Clears, *IEEE Transactions on Knowledge and Data Engineering,* (December 1989).

8. Herlihy, M., and Wing, J., Avalon: Language Support for Reliable Distributed Systems, in *Proceedings of the 17th International Symposium on Fault-Tolerant Computing,* (July 1987) pp. 13—26.

9. Hudson, Scott E., and King, Roger, Object-Oriented Database Support for Software Environments, in *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, May 1987) pp. 491—503.

10. Hudson, Scott E., and King, Roger, The Cactis Project: Database Support for Software Environments, in *IEEE Transactions on Software Engineering,* **914, 6** (June 1988) pp. 709—719.

11. Kaiser, Gail E., and Feiler, Peter H., An Architecture for Intelligent Assistance in Software Development, in *Proceedings of the 9th International Conference on Software Engineering,* (March 1987) pp. 180—188.

12. Kaiser, Gail E., Barghouti, Naser S., Feiler, Peter H., and Schwanke, Robert W., Database Support for Knowledge-Based Engineering Environments, in *IEEE Expert,* **3, 2** (Summer 1988) pp. 18—32.

13. Kaiser, Gail E., Feiler, Peter H., and Popovich, Steven S., Intelligent Assistance for Software Development and Maintenance, in *IEEE Software,* (May 1988) pp. 40—49.

14. Kim, W., Banerjee, J., Chou, H.-T., Garza, J., and Woelk, D., Composite Object Support in an Object-Oriented Database System, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1987,* (Orlando, Florida, October 1987).

15. Lindsay, B., McPherson, J., and Pirahesh, H., A Data Management Extension Architecture, in *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, May 1987) pp. 220—226.

16. Linton, Mark A., Queries and Views of Programs Using a Relational Database System, in *Ph. D. Dissertation,* (University of California, Berkeley, December 1983).

17. Linton, Mark A., Distributed Management of a Software Database, in *IEEE Software,* (November 1987) pp. 70—76.

18. Liskov, B., and Scheifler, R., Guardians and Actions: Linguistic Support for Robust, Distributed Programs, *ACM Transactions on Programming Language and Systems,* **5, 3** pp. 382—404.

19. Morrison, Ron, Brown, Fred, Connor, Richard, and Dearle, Al, The Napier88 Reference Manual, *Persistent Programming Research Report PPRR-77-89,* (Universities of Glasgow and Saint Andrews, July 1989).

20. O'Brien, Patrick D., Halbert, Daniel C., and Kilian, Michael F., The Trellis Programming Environment, in *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications 1987,* (Orlando, Florida, October 1987) pp. 91—102.

21. Paul, H., Schek, H., Scholl, M., Weikum, G., and Deppisch, U., Architecture and Implementation of the Darmstadt Database Kernel System, in *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, May 1987) pp. 196–207.

22. Reiss, Steven P., Connecting Tools Using Message Passing in the *FIELD* Program Development Environment, *IEEE Software,* (to appear, 1989).

23. Richardson, Joel, and Carey, Michael, Programming Language Constructs for Database System Implementation in EXODUS, *Proceedings of ACM SIGMOD 1987,* (San Francisco, California, May 1987) pp. 208–219.

24. Rowe, Lawrence A., and Wensel, Sharon, Editors, Proceedings of the 1989 ACM SIGMOD Workshop on Software CAD Databases, in (Napa Valley, California, February 1989).

25. Stroustrup, Bjarne, The C++ Programming Language, Addison-Wesley (1986).

26. Teitelman, Warren, and Masinter, Larry, The Interlisp Programming Environment, in *IEEE Computer,* **14, 4** (April 1981) pp. 25–33.

27. Wile, David S., Carving Up SCAD Databases, in *Proceedings of the 1989 ACM SIGMOD Workshop on Software CAD Databases,* (Napa Valley, California, February 1989) pp. 5–8.