A Model for Computer Life

Witold LITWIN Gio WIEDERHOLD

Stanford University, Stanford CA 94305[1]
Hewlett Packard Laboratories, Palo Alto

Abstract

We propose to organize the logical level of the computers interconnected worldwide as a kind of a life modeling our own. We assume this universe populated with programs of a new kind called beings. Beings do not need to be intelligent, their goal is to provide services through some internal and external capabilities. They are autonomous, active and interoperable to the point to behave living individuals. They are aware of their capabilities, able to use those of other beings, may acquire new capabilities and may give birth to new beings. We show that the model is feasible already and that beings may be implemented as multidatabase systems, storing data and capabilities and working in a federation. We discuss perspectives for the software engineering and the database systems that follow from the model.

Motto. *From the "Popular Journal", 1860 : Well informed people know that it is impossible to transmit voice over wires and that were it possible to do so, the things would be of no possible value.*
Historical note. *The telephone was invented in 1861. Shortly after, President of the Western Union Telegraph Comp. turned down exclusive rights to Bell's invention, explaining "What use could this company make of an electrical toy ?".*

1. INTRODUCTION

1.1. Evolution of data management goals

Computers will soon be interconnected worldwide. They will have database software to serve the sharing of large data banks [Cod70]. The evolution of the technical development was as follows :

- Data to be shared were removed from autonomous files and redefined as an integrated collection called a database [Dat86]. The collection was to be under central control of the database administrator, on a large mainframe.

- To improve access performance, the database was proposed to be distributed over a number of interconnected sites [Chu87], [Cer87]. The system was expected to provide logically central management over distributed data. The data were allowed to be contained in multiple autonomous databases. Now the system should provide functions for non-procedural manipulations of data in different databases [Lit87]. Data were not assumed to be integrated, as administrators should have the autonomy of naming, choosing the data structures and values according to their needs [Lit82], [Hei85], [Tem87], [Kuh88], [Jac88], [Rus88], [Wol89].

The latter type of systems is called multidatabase systems, or federated systems [Lit82], [Hei85], [Day85], [Sar87]. Databases that are jointly manipulable without integration are called interoperable [Lit86], [Int87]. Research on interoperability showed that it is useful for a multidatabase system to interoperate not only with database systems, but also with other types of systems. For instance, to update dynamic attributes, the multidatabase system MRDSM calls the formal calculus system MACSYMA. There is no question to integrate MACSYMA under MRDSM, MACSYMA is several times larger than MRDSM [Lit87b].

1.2. Remaining drawbacks

The notion of using services of different systems, leads to the idea of interoperable systems, which are autonomous systems that may be manipulated together without integration. Such systems are the next step beyond the multidatabase systems [Lit89]. However, if designed according to the current principles of software engineering, the interoperable systems will still leave a significant burden to the users:

- Computers carry a skyrocketing amount of software. The classification of this software is now well understood and falls into well understood classes of services such as: database systems [Bur86], text editors, compilers, image manipulators,... The programs realizing these functions duplicate a large number of components realizing subfunctions or services [Gas87]. On the other hand no program provides everything that might be needed for an application. Also, users are generally lost in subtle differences between programs with similar purpose.

- Systems are now being designed to provide user autonomy [Abb88], [Gar88]. However, these systems are designed for what the author imagined the user wants, not less (unless they have bugs) and not more. They have no autonomy with respect to their designer and user, and behave in a predetermined way. If an unexpected function is needed, the designer has to rewrite the program. The user

[1] Witold Litwin is on leave from INRIA & Univ. Paris 9, France, litwin@eclipse.stanford.edu, wiederhold@sumex-aim.stanford.edu

has to wait or to find another program with the desired function, which may lack in turn functions the user had available.

1.3. A solution

We need a model to organize the computer universe. We therefore sketch such a model and call it *computer life.* Its axiom is that *the ultimate goal of our activity inside the computer universe is to set up a kind of a life, mirroring ours*. To organize it, we should transpose the rules and properties that proved most effective for ourselves. Software designers already do it frequently intuitively, but only for particular applications.

To create the computer life, we propose to populate the computer universe with programs of a new kind we call *beings.* Beings are autonomous, active and interoperable to the point they behave largely like us in the professional life (but they are anthropomorphic creatures for other aspects of our lives). They survive in general only if they generate profit covering their living expenses. They are aware of their capabilities to provide services and may interoperate with other beings. They do not need to be intelligent, just skilled. They may self-modify and may give birth to new beings. They handle their decisions usually autonomously without making us aware of. They may also create organizations providing services beyond capabilities of a single being. While these organizations are primarily for their own benefit, they mirror organizations like banks, hiring agencies, schools, law enforcement institutions,... [Lit89a].

We argue that the computer life model is a rewarding organization for ourselves as well. We should have less hassle in managing the computer universe. We will also use operational capabilities of beings that they could create or improve by themselves. Finally, we may gain financial profit from the lease of the services of a being. The self-organization characterizing the computer life looks also the only way for the computer universe to reach the scale of millions of interconnected nodes we will require soon.

We also argue that the model is not a science fiction. While almost not similar work exist, the knowledge objects KNOs [TsI87] present some properties of beings, as well as the active objects with learning capabilities [McL88], the actors [Hew86], and the agents with beliefs and desires [Tho89]. Also, the capabilities of beings called user agents, representing our interests inside the computer universe are these of mediators [Wie89]. More generally, the state-of-the art of the tools like multidatabase systems and database systems, object oriented paradigm and logic programming, makes the model already feasible. We show how these technologies let us to put the computer life into practice.

We finally show that even a partial implementation of properties foreseen for the beings leads to interesting short-term gains for the software engineering and the database systems. We focus on the idea that a (multi)database system implementing a being manages both data and capabilities. We show that application programs could be generated by queries. They become open ended, and able to provide services they were not designed for. The frontier between application programs and databases disappears, and more flexibility results from for the users.

Section 2 defines the model. Section 3 discusses details and the feasibility of the model. Section 4 shows the perspectives for the software engineering and the role of database systems. Section 5 contains the conclusion.

2. THE MODEL

2.1. The basis

The model has for goal the organization of the computer universe at the logical level. Physical details, like size of memories, type of computer used or interconnection procedures are irrelevant. The model starts from the observation that the main phenomenon organizing our universe is the life. Its remarkable property is that it is self-organizing and continuously evolving towards a more effective organization. To set up a kind of a life in the computer universe should therefore be the most effective way to organize this universe as well. This is the main axiom of the model. We believe that this is the goal of our activities concerning this universe already.

A life is seen as an organization of a collection of individuals. The individuals seek gain exceeding expenses for survival. The organization also lets the collection to persist, despite the death of the individuals that do not survive. The individuals are autonomous and they interoperate. They also exhibit an active and adaptive behavior to survive. Finally, they are able to create new individuals to maintain their life forever.

The major aspect of life pertinent to the model is that an individual provides services to others. The benefits in the computer life come from such activities. The model is the framework for organizing the computer universe towards this goal. The general approach is to map appropriate aspects of our life. Many rules to organize efficiently any universe, are indeed already in our life. We now outline the principal concepts. More details are in [Lit89a].

2.2. Beings

A *being* is an individual inside the computer universe. It is a new type of program that is a tightly coupled, autonomous and active collection of capabilities oriented towards service to other beings. It is not necessarily intelligent, in the sense common to the artificial intelligence, it is just skilled. The overall effectiveness of a being is measured by the difference between the gain from the services the being provides and its expenses for staying alive. The capabilities themselves are programs that are not autonomous.

Formally, a being B is a triplet $B = (Co, V, Cs)$. Co are the *operational capabilities* of the being through which it provides the services. V (Curriculum Vitae) is a self-description of B, to make other beings aware of B's capabilities. Cs denotes the *survival capabilities* that allow B to survive and, especially, to find the way to earn more than it spends and to create new beings. Survival capabilities also allow B to autonomously decide to accept or refuse a task, to exchange information with other beings, to seek for employment etc. A being that looses more than it gains usually dies.

The birth of a being may result of our action. It may also result of an individual act of a being (cloning), or from a cooperative act, basically between two beings. The child inherits some, but not necessarily all the capabilities of the parent(s) [Lit89a].

The crucial aspect of the autonomy of a being is that it provides capabilities that were not built-in. For this purpose, the being may invoke external capabilities, like we use various tools. It can also make requests for service to other beings. Finally, it can import capabilities into itself for durable usage. Both *Co* and *Cs* sets of capabilities are thus time-dependent.

2.3. Organizations

Beings may form organizations providing more services than an individual being. An organization *O* is formally a quadruple $O = (Co, V, Cs, \mathbb{B})$. $\mathbb{B}$ denotes the time-dependent set of beings forming *O* (employees of *O*). Any capability of *O* is that of an employee or is a composition of capabilities of it employees.

The difference between an organization and a being is that the constituents of a being (capabilities) are not autonomous. An organization may also provides resources to the employees.

As beings, organizations may acquire capabilities that were not built-in and drop the useless ones. This is done through the evolution of the employees, and the hiring or firing of the employees. The evolution may be on the initiative of an the employee or may be requested by the organization. The corresponding decisions are autonomous with respect to us, ie, we are not aware of them.

The computer universe will need a number of organizations useful for our own life. There should be hiring agencies, journals and universities for capabilities exchange and diffusion. There should be travel agencies to let the beings to move elsewhere. One will also need hospitals for damaged beings, courts, judges, police, etc. Not all beings will be gentle, as they may be set up by humans who are not [Cac89].

2.4. Autonomy, interoperability and competition

A being should find operational capabilities it needs and does not have without our intervention. These capabilities are provided by other beings or organizations. The being should choose them on competitive basis. It should sell its own capabilities (or of its organization) to other services (beings or organizations). The curricula vitae *V* exist precisely for this purpose. Protocols for interoperability and operational capabilities should allow a being to exhibit*V* to others and to change it with the time. A being should also be able to to negotiate prices of services it provides or requests. In this sense, a being is aware of its existence, as we are aware of ours.

To provide a service, a being or an organization receive a request for. A request may be thought of as a high level message to an object or an actor [Hew86]. The properties of a being make however this concept richer than that those two, going beyond that of an active object [McL88]. Objects and actors are basically passive with respect to beings and present a much lower degree of autonomy and interoperability. These concepts do not have a clearly defined objective of curricula, do not require ability to self-change the set of capabilities, to give birth to other objects or to create organizations. An exception are the knowledge objects (KNOs [Tsi87]). The concept of a KNO is the closest to that of a being, though is more restrictive. Beings are not required to communicate only through black-boards, a KNO does not carry a vitae etc.

2.5. Interface to our universe

Some beings will receive request for service from human users. They will provide these services by themselves or will issue requests to other beings. Such beings will be *user agents* , extending the current meaning of this concept. A *marionette* KNO, is an example of a user agent. A user agent will however usually have capabilities of a *mediator* [Wie89], presenting thus a more autonomous behavior. A user agent may also belong to a user who is then its *owner*. The owner has the control of its agent, priority for service and (especially) financial benefits from the services dispensed by the agent.

The degree of the autonomy of a being means that its behavior will be largely undeterministic with respect to this of the present programs. We will progressively lose control of the computer universe that will rely upon itself. An indirect benefit is a reduction of hassle in managing this universe. The self-organization of the computer life is probably the only way to manage millions of interconnected nodes we will have soon.

The direct benefit of the existence of beings is two fold. We will utilize their operational capabilities, or will earn money for services of "our" beings to others. This money may directly enter a being owner's accounts in the computer universe. Alternatively, an agent may have its own accounting system, the owner's part being collected through some tax collection system. An owner may have several agents and may be an organization.

It is worthy to note that the owners will appear from the computer universe as a kind of Greek Gods. There will be multiple Gods with autonomous conflicting interests and sometimes evil intentions. Fights between God's will become these of their agents.

3. FEASIBILITY OF THE MODEL

3.1. Overall analysis

While the model may appear as science fiction, many pieces exist already. The services and capabilities may be described using the abstract data types or object oriented principles. Rules to be defined for various organizations to create, may be implemented using the logic programming. The production, evolution and the mutual understanding of curricula vitae between beings may be achieved through the self-description [Rou83], [Rou85], [Ccs87], [Bat88]. Efficiency of cooperation between distributed services may be achieved through the usage of techniques for parallel and distributed processing of autonomous data. They are now under intensive studies [Alo87], [Bre87], [Bel87], [Dee87], [Elm87], [Wie87], [Pu87]. The technique of contract nets [Smi80] seems particularly adapted to the kind of negotiations a being will be usually involved in. Incidentally, it is an instructive example of an intuitive application of the model.

The model adds to these paradigms the goals of autonomy and of interoperability at all levels. It also stresses the fundamental importance of these goals which only start to be fully appreciated [Eli87], [Dai88], [Gar88]. While many technical problems remain to be solved, it is also promising that the model principles worked already rather well for ourselves.

To examine more in detail the feasibility of the model functions, we now review an example. We then focus on

some concepts transposed from our own life that are useful for the computer universe as well. We analyze their feasibility and we point out research issues.

3.2. Example

The user agent will usually reside on the user workstation. It is likely that the workstations will be individually owned and will be permanently a part of the computer universe connected through the Open System Architecture [Hew85], [Iso87], [Dai88]. A workstation may be the location of the number of services and of beings, only some of them belonging to the workstation's owner. The composition of the services on the workstation will probably be decided by the user agent or the mediator [Wie89].

Consider a user wishing to write a text. The agent may itself have the corresponding capabilities ie may be a *writer* (a sophisticated word processor in the current terminology). If not, it will call for a writer. The writer may be on the workstation or elsewhere. A distant writer may generate a clone that will travel to the workstation.

During the writing, it may happen that the user requests a capability that the writer does not have, such as to solve an equation. The writer attempts then to find this capability elsewhere. It either posts a request for a mathematician (equation solver in the current terminology) to delegate him the whole task or finds the capability somewhere and either applies it or even imports it into itself. In the latter case the capability may be dropped later on or kept permanently. The mathematician may move to the writer's workstation or may entirely or partly copy itself to the workstation as a clone. Alternatively, the writer may send the equation to the mathematician or to the organization the mathematician works for.

To gain money, for its survival and/or for its owner, if any, the writer also seeks to sell its services to other beings. If idle, it is supposed to use the time to hunt for new capabilities to enhance its profitability. It may also advertise its services somewhere. It may finally do some housekeeping, update its vitae, etc.

3.3. Implementation of beings

A practical kernel for the implementation of a being may be a dedicated multidatabase system (MBS), especially a relational one. A being will consist of the system itself and of some databases under its management. The kernel and its databases will be identified by the multidatabase name that would be the logical name of the being itself. The commands of the multidatabase language of the system, like these of MSQL [Lit87] or of VIP-MDBS [Kuh88], will allow merging in a single query data from different databases. They will also provide the interoperability with respect to data in the autonomous databases of other beings. Other features characteristic of an MBS or of a DBS will allow for concurrent processing, for privacy, data security, etc. These features will be the kernel for the survival capabilities of the being.

A new fundamental feature of some of these databases should be a POSTGRES like ability to have attributes whose values would be the capabilities themselves [Sto86]. A command analogous to EXECUTE command in POSTGRES, should allow to execute selected capabilities. We call this command APPLY. We assume that APPLY has one selection expression for capabilities and an optional selection expression for data it applies to. The latter expression follows then the keyword TO, to be placed after the former expression. The clause TO is optional as the capability may find its data itself.

Basically, the system activates in parallel all the selected capabilities, although they may determine the execution order by themselves. Especially, a capability should be able to call another one or even itself recursively. The flexibility of the multidatabase manipulation language of the system will allow import and export of the capabilities. It will also allow dropping of useless ones. Finally, it will allow to define complex operations, involving several beings, through multidatabase transactions.

Other databases will contain data relative to the vitae. These data and the corresponding needs are discussed below. Finally, some databases may be necessary as the working environment for the capabilities.

Example. The writer could bear the logical name, let us say John Smith or John in short. Its kernel MBS may have a database named **Op-Capabilities** with the following relation :

Capab (Name, Type, Source, Version, Code)

where **Name** is the name of a capability, **Type** its type, eg, Speller, **Source** its source or supplier like Microsoft, **Version** is the version id., and **Code** is the code to be executed. To check the spelling of a word, John may issue to its kernel the query :

```
USE Op-Capabilities
APPLY Code FROM Capab WHERE Name = 'Speller'
```

Once in control, the speller will then ask John for the word to check. If it should be applied to a whole text, somewhere in a database, then a TO clause would follow the WHERE clause above and would bring the text.

A simple request for service to John from Nick could be:

```
USE John.Op-Capabilities
APPLY Code FROM Capab WHERE Name = 'Speller'
```

To provide the service, John would allow the query to be executed by its MBS. IF John wants Nick to learn this capability, then John will issue the interdatabase query :

```
Use John.Op-Capabilities Nick.Op-Capabilities
INSERT INTO Nick.Op-Capabilities . Capab
SELECT * FROM John.Op-Capabilities . Capab
WHERE Name = 'Speller'
```

From now on, Nick will dispose of the capability by its own.

3.4. Curriculum Vitae

The Curriculum Vitae *V'* is a fundamental tool for the interoperability of the beings. *V* should carry information allowing to evaluate its bearer skills. The evaluation should

be performed autonomously, by the beings themselves. A vitae should contain the following parts.

3.4.1. Definition of the capabilities

Many programs have capabilities described today through the form of an explicit menu or of icons. Such a description usually rely on some implicit capabilities. These are not defined precisely. We need a language defining capabilities and their expression through other capabilities in some standard form. This description should form the *capability description* section of*V*. The description should be oriented towards the usage by the beings. It should be evolutive, as the being can gain new capabilities.

A language may consist of standard names of capabilities, eventually with parameters. Abstract data type approach and object oriented languages and systems are more elaborated candidates. A starting point may also be the capability description language like in [Rya86]. One should also consider the work on the self-description of databases [Rou82], [Rou83], [Rou85], [Mar87].

Anyhow, it does not seem practical to require the full description of each capability in each *V*. One should consider external databases defining in depth capabilities. *V* should usually contain only the generic definitions, defining together the *profile* of the being.

3.4.2. References

The process of evaluation of capabilities should also be autonomous. A helpful tool may be the *reference* section in *V*. This section should describe the work experience of the being. One issue is the language for the description of this information. While the general solution is an open issue, a lot may be achieved if the kernel of the being is an MBS.

3.4.3. Salary

A being should generate profit. Its CV should indicate the price of its services, in a *pricing* section. The pricing should be established basically by the being itself. An interesting issue is how it can be done.

3.4.4. Interview

The generic capabilities announced in *V* may be not understandable to another being. Also, the confidence in the *V* should be limited. The being examining *V* of another one should be able to interview its bearer. This may consist of requests for details of the bearer's capabilities. It may also include the benchmarking of corresponding performance. The interview may be done by a specialized being or organization.

3.5. Organizations

The beings that formed an organization should pool their capabilities for the group work. This leads to a number of technical issues. We outline a few:

- joint representation as an organization.
- choice of the most effective manner to perform the work.
- internal structure of the organization and its its conceptual scheme.
- internal communication and management of various dependencies [Mar85].
- some kind of legal responsibility, with respect to the privacy and security of the universe.
- negotiation protocols between the beings and organization.

The proposed way to implement the beings may help to achieve these needs. The negotiation protocols may be an extension of those for federated databases [Hei85].

3.6. Exchange of capabilities

It is assumed that some beings export capabilities. In general a being also uses external capabilities. There are following ways for it :

(a) - the being calls another being or organization. This being or a representative of the organization executes then the task requiring the capability on the behalf of the requestor. This is the groupware approach.

(b) - the being learns through some script how to use capabilities available in some recipients, database or knowledge base. These capabilities remain outside the being. It only uses them as we use tools helping us to do a task.

(c) - the being learns the capability which means that it imports the corresponding code into itself.

An elementary implementation of (a) is to use remote procedure calls. The requester knows then who provides the service. If this information is unknown, one may design a mediator describing who provides a given service, how much it costs etc. Services may then be searched for using database or knowledgebase queries.

This type of cooperation will require also standards for the interfaces between capabilities. Many such de facto standards exist already, especially for microcomputer software. There are many independently developed auxiliary programs in this area, able to interoperate over data in format of some basic programs. For instance, there are several speller checkers that operate on the text of MsWord while the text is being written, as if they were parts of the MsWord.

In the case (b), the being will need to purchase the right to use capability from some server. It may then get a script (transaction scheme) to be downloaded it into its private database. Like a manual, the script will define how to use together local and server's capabilities, what data should be sent to the server etc. To use the server may be advantageous if the server is on a powerful computer, or more up to date.

However, the strategy (c), that is importing a capability, may lead to similar advantages, and offers better control over the capability. The implementation of this strategy may require nevertheless software engineering techniques, we are not aware of presently. However, while one may foresee technical problems related to the transition between modified source code, new object code, linking etc., none of them seem really hard to solve. One basis for (c) may be techniques for extendible database systems, if they are generalized to make the extensibility autonomous.

On the other hand, the use of an MBS as a kernel, also may allow to realize the goals (a) to (c)., as the following example shows.

Example. Assume that CompuServe server has the database **Capabilities** with the following scheme :

Capab (Name, Version, Code, Interface, Type)

Cap_pricing (Name, Rent, Buy&Share, Buy&Copy, Update)

Usage (User-Id, Name)

Volume_Discount (User-Id, Name, Discount).

To download cheap equation solvers, the writer John may issue the following MSQL query with obvious meaning:

```
USE CompuServe.Capabilities
        John.Op-Capabilities
INSERT INTO  Op-Capabilities.Capab
LET X BE CompuServe.Capabilities
SELECT X.Capab.Name X.Capab.Version  X.Capab.Code
        X.Capab.Type
FROM  X  John.Op-Capabilities
WHERE X.Cap_pricing.Buy&Copy < 10 $   AND
        X.Capab.Type = 'Equation solver' AND
        X.Capab.Name = X.Cap_pricing.Name
```

4. IMPACT ON SOFTWARE ENGINEERING

While the concept of a being with all its properties is futuristic, we have shown hopefully that it is feasible. In [LIT89a], one finds further discussion of the beings. Below, we show that an implementation of the property that a being is a database system (DBS) able to manipulate data and capabilities in a multidatabase environment should already have an impact on software engineering and on the role of database systems.

4.1. Architecture of programs

A classical program is a set of instructions of fixed size and with predefined set of possibilities, responding to also predefined user needs. It is not updatable once compiled and link edited. In contrast, data in a database, may be modified, change the size and may be retrieved in many ways to satisfy unexpected needs. The concept of a being transfers these properties also to programs.

As the result, the notion of a program changes, at least for the application programing. A program becomes a set of capabilities dynamically defined by a query. This program has neither a fixed size nor predefined possibilities. These properties are desirable for programs as they were for data.

On one hand, a program may be asked to perform an unexpected task, as in examples above. For a task to be reexecuted, it may apply a new version of a capability that was transparently improved or may select a competing capability etc. Clearly, a yet unknown degree of flexibility becomes available to the users. While such a flexibility was not that important in the classical environment of application programs and of batch processing, it is fundamental for modern interactive users.

To design a program becomes mainly the problem of an associative retrieval of capabilities and of communication between them. (Multi)database languages and systems are now powerful enough to allow selections of extremely various set of capabilities. The control statements of an application program may be designed as some capabilities as well. This approach may deal better with complex decision sequences through a shared use of decision tables, or of guarded commands or of knowledge-based techniques, as proposed in [WIE82]. A new problem for the software engineering are the techniques for the design of capabilities. Some issues will be briefly addressed below.

4.2. Role and architecture of a database system

A DBS may consist itself of capabilities and of a bootstrap program that would initialize the DBS once it is called. The bootstrap would load or at least would activate a number of capabilities own to DBS, necessary for query decomposition, transaction management, etc. The choice of these capabilities may depend on the machine, the working environment, the task itself, etc. The bootstrap or a capability it activates may act as the system generator in mainframe operating systems.

A DBS may now be viewed as an operating system of a new type. The main difference is the use of an assertional language to select tasks, instead of simple commands (as well as the data manipulation capabilities of a DBS). The benefit is a new flexibility and open-ended possibilities for the user, including access to capabilities of other DBSs.

A user of DBS may wish not only to import or export existing capabilities, but also to design capabilities himself. The DBS should provide an adequate programming environment. Although, the corresponding functions are new for a DBS, they are basic to the software engineering. Note that DBS should be more protective of errors in capabilities than of those in data.

A frequent view of a DBS is that it is a repository of data for the application programs, ie, a kind of persistent data storage. The role of a DBS becomes larger in the proposed approach, Not only data, but also application programs come under its control and attractive possibilities appear. It is like a revenge of database methodology over the methodology of programming languages.

4.3. Interoperability of capabilities

It is desirable to allow capabilities to be created by autonomous designers. They have therefore to be interoperable. The federated database approach develops the corresponding principles for data access. The application of these principles to capabilities could be as follows:

- the capabilities read and write only the logical data managed by the DBS. They use for this purpose the data manipulation language of the DBS. In other terms, only DBS is in charge of sharing data among capabilities.

- The management of physical data structures is the task of DBS. The capabilities should neither deal with the corresponding optimization issues, nor with the concurrency control, recovery etc. Unlike classical programs, capabilities should not pass to each other information at a low level data: pointers, physical location of variables etc.

- capabilities may need to deal with autonomous data. Through the DBS language one should be able to deal with the corresponding issue of name, value type and data structure autonomy. A DBS should be able to convert data, if the data types exchanged between capabilities differ (eg, by units of measure).

- the description of capabilities may be to some extent heterogeneous. Again, the DBS language should allow to deal with mismatches.

- structures hidden to the DBS, such as a stack for instance inside a complex attribute, should be avoided. They may however be necessary, if DBS does not provide an alternative choice.

- for import/export, the capabilities themselves should be defined in a high level language. This language should be understandable by all DBSs in the federation, as is the

common data manipulation language in the federated systems. It probably should even be a part of it. It should depend on the local DBS, whether the capabilities are stored in this language or in some kind of object language resulting a compilation.

- despite the common description, it will happen that some capabilities will need more resources than a computer with a given DBS is able to provide. It will be a responsibility of a DBS to decide whether a foreign capability should be imported or the corresponding data should be exported. The user query will need to be optimized accordingly.

5. CONCLUSION

The principles of autonomy and of interoperability lead to the organization of the computer universe as a life modeling our own. Living individuals are then programs of a new type called beings that are more general and autonomous. Like us, they seek to survive providing and using services, may import and export capabilities, may multiply and may exhibit to other beings the description of their skills and of their experience. They are also able to create organizations.

The computer life model appears attractive and feasible, although its creation is a major enterprise. Multidatabase systems able to execute capabilities appear at present the best tool. They open attractive perspectives for database systems and software engineering.

While we have surveyed the applicability of the model, we did not enter into details, typical of a research contribution. It is nonetheless also a role of researchers to introduce new frameworks. Specific contributions will find in the model their overall purpose. The perspectives are vast and fascinating.

Acknowledgments

We thank Nick Roussopoulos and Dennis McLeod for many suggestions.

This work was supported at Stanford University by DARPA contract N39-84-C-211 (task 24) on Knowledge-based Data Management. It is largely based on [LIT89], supported by the Institute of Advanced Computer Studies of the University of Maryland (UMIACS), by the National Science Foundation under Grant CDR-85-00108 and by the Institut National de Recherche en Informatique et en Automatique (INRIA) under the INRIA - SRC cooperation protocol.

REFERENCES

[Abb88] Abbott, K. R., McCarthy, D. R. Administration and Autonomy in A Replication-Transparent Distributed DBMS. 14-th Int. Conf. on Very Large Databases, Los Angeles, USA, (Aug. 1988), 195 - 205.

[Alo87] Alonso, R., Garcia-Molina, H., Salem, K. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 5-11.

[Bat88] Batini, C. Di Battista, G. A methodology for conceptual documentation and maintenance. Inf. Syst. 13, 3, 1988, 297-318.

[Bel87] Bellcastro, E & all. DQS -Distributed Query System. (Sept. 1987), CRAI, Italy, 21. Int. Conf. on Extending Database Technology, Springer Verlag, 1988.

[Bre87] Breitbart, Y., Silberschatz, A., Thompson, G. An Update Mechanism for Multidatabase Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 12-18.

[Bur86] Burns, T. Fong, E. Jefferson, D. Knox, R. Reedy, C Reich, L. Roussopoulos, N. Truszkowski, W. Reference Model for DBMS Standardization. ACM SIGMOD Records, (March 1986).

[Cac89] Special Section on Internet Worm. CACM, 32, 6 (June 1989), 677-710.

[Cod70] Codd, E., F. A Relational Model of Data for Large Shared Data Banks. CACM, 13, 6, 1970, 377-387.

[Ccs87] Consultative Committee for Space Data Syst. : Standard Formatted Data Units - Structure and Construction Rules. Red Book, Issue 2, (Feb. 1987). Nat. Aeronautics and Space Adm.

[Cer87] Ceri, S., Pernici, B., Wiederhold, G. Distributed Database Design Methodologies. Proceedings of the IEEE, (May 1987), 533-546.

[Chu87] Special Issue on Distributed Database Systems. Chu, W. (ed). Proceedings of the IEEE, (May 1987), 532-735.

[Dai88] Distributed Aspects of Information Systems (DAISY Working Group Rep). Research into Networks and Distributed Applications. R. Speth (ed.). Elsevier Science Publ. 1988, 1029 - 1049.

[Dat86] Date, C., J. An Introduction to Database Systems. 4-th Ed. Vol. 1. Addison-Wesley, 1986, 639.

[Eli87] Eliassen, F., Veijalainen, J. Language Support of Multi-database Transactions in a Cooperative, Autonomous Environment. IEEE Region 10 Conf., Seoul, (Aug. 1987).

[Elm87] Elmagarmid, A., Leu, Y. An Optimistic Concurrency Control Algorithm for Heterogeneous Distributed Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 26-32.

[Fan88] Fankhauser, P., Litwin, W., Neuhold, E., Schrefl, M. Global View Definition and Multidatabase Languages : Two Approaches to Database Integration. Research into Networks and Distributed Applications. R. Speth (ed.). Elsevier Science Publ. 1988, 1069-1082.

[Gar88] Garcia Molina, H., Kogan, B. Node Autonomy in Distributed Systems. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 158-166.

[Gas87] Gash, B., Kelter, U., Kopfer, H., Weber, H. Reference Model for the Integration of Tools in the "EUREKA Software Factory". ACM-IEEE Fall Joint Comp. Conf. (Oct. 1987), 183-190.

[Ham79] Hammer, M., McLeod, D. On database management system architecture. MIT Lab. for Comp. Sc. MIT/LCS/TM-141, (Oct 1979), 35.

[Hei85] Heimbigner, D., McLeod, D. A Federated Architecture for Information Management. ACM Trans. on Office Information Systems. (July 1985), 3, 3, 253-278.

[Hei87] Heimbigner, D. A Federated System for Software Management. IEEE Data Engineering, (Sep.1987), 10, 3, 39-45.

[Hew85] Hewitt, C., De Jong, P. Open Systems. On Conceptual Modeling. Springer Verlag, 1985, 147-164.

[Int87] Interoperable Database System. 1st International Symposium. INTAP, (May 1987), 167.

[Iso87] Remote Database Access Protocol. 2-nd Working Draft. ISO/TC 97/SC 21/WG 3, 1987.

[Jac88] Jakobson, G., Piatetsky-Shapiro , G. Lafond, C., Rajinikanth, M., Hernandez, J. CALIDA : A Knowledge-Based System for Integrating Multiple Heterogeneous Databases. 3-rd Int. Conf. on Data and Knowledge Bases : improving usability and responsiveness. Jerusalem, (June 1988), Morgan Kaufmann Publ., 3-18.

[Kuh88] Kuhn, E., Ludwig. Th. VIP-MDBS : A Logic Multidatabase System. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 190-207.

[Li87] Li, Q., McLeod, D. Object Flavor Evolution through Learning in an Object-Oriented Database System. 2nd Int. Conf. on Expert Database Systems. The Benjamin/Cummings Publ. Comp. 469-495.

[Lit82] Litwin W. et al. SIRIUS Systems for Distributed Data Management. Ed. H. J. Schneider. North-Holland, 1982, 311-366.

[Lit86] Litwin W., Abdellatif, A. Multidatabase Interoperability. IEEE Computer, (Dec. 1986), 19, 12, 10-18.

[Lit87] Litwin W., et al. MSQL : a Multidatabase Language. Inf. Science - An International Journal, Special Issue on Databases, 48, 2 (July 1989).

[Lit87b] Litwin W., Vigier, Ph. New Functions for Dynamic Attributes in the Multidatabase System MRDSM. HLSUA Forum XLV, New Orleans, (Oct. 1987), 467-475.

[Lit89] Litwin W., Mark, L. Roussopoulos, N. Interoperability of Multiple Autonomous Databases. System Research Center. Univ. of Maryland, College Park. Techn. Rep. TR 89-12, 45.

[Lit89a] Litwin W., Roussopoulos, N. A Model for Computer Life. Res. Rep. UMIACS-TR-89-76, University of Maryland Institute for Advanced Computer Studies, (July 1989), 20.

[Mar85] Mark, L., Roussopoulos, N., Chu, B., Update Dependencies. IFIP TC2 WG 2.6 Working Conference on Database Semantics, (Jan. 1985), Belgium.

[Mar87] Mark, L. Roussopoulos, N. Information Interchange between Self-Describing Databases. IEEE Data Engineering, (Sep. 1987), 10, 3, 46-52.

[Mar87a] Mark, L., Roussopoulos, N. Operational Specifications of Update Dependencies. SRC Res. Rep., Univ. of Maryland, (Feb. 1987), 44.

[McL88] McLeod, D. A Learning-Based Approach to Meta-Data Evolution in an Object-Oriented Database. Advances in Object-Oriented Database Systems. Springer-Verlag Lecture Notes in Comp. Sc., 1988.219-224.

[Pu87] Pu, C. Superdatabases : Transactions Across Database Boundaries. IEEE Data Engineering, (Sep. 1987), 10, 3, 19-25

[Rou84] Roussopoulos, N. Mark, L. Update Dependencies in Relational Databases. 1st International Conference on Expert Database Systems, Kiawah Island, South Carolina, (Oct. 1984).

[Rou85] Roussopoulos, N. Mark, L. Schema Manipulation in Self-Describing and Self-Documenting Data Models. Int. J. of Comp. and Inf. Sc., 14, 1, 1985, 1-28.

[Rus88] Rusinkiewicz, M et al. Query Processing in OMNIBASE : a loosely coupled multi-database System. Tech. Rep. #UH-CS-88-05, Univ. of Houston, (Feb. 1988), 27.

[Rya86] Ryan, K., Larson, J. The use of E-R models in capability schemes. 5-th Conf on E-R Approach. Dijon, France (Nov. 1986).

[Sam88] Samy Gamal-Eldin, M., Thomas, G. Elmasri, R. Integrating Relational Databases with Support for Updates. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 202 - 209.

[Sar87] Special Issue on Federated Database Systems. Sarin, S. (ed.). IEEE Data Engineering, (Sep. 1987), 10, 3, 64.

[Smi80] Smith, R. G. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. IEEE TC, C-29, 12, (Dec 1980), 1104-1113.

[Sto86] Stonebraker, M. Rowe, A. L. The Design of POSTGRES. ACM-SIGMOD 86, 340-355.

[Tho89] Thomas, B. Shoham, Y. Schwartz, A. Modalities in Agent-Oriented Programming (prel. report), Stanford University, Computer Forum 1989.

[Tsi87] Tsichritzis, D. Fiume, E., Gibbs, S., Nierstrasz, O. KNOs: KNowledge Acquisition, Dissemination, and Manipulation Objects. ACM-TOIS, 5, 1, (Jan 1987), 96-112.

[WIE82] Wiederhold, G. Scientific Computing. Roles of Industry and the University in Computer Science Research and Development. National Academy Press. 1982, 60-66.

[Wie87] Wiederhold, G., XiaoLei, Q. Modeling Asynchrony in Distributed Databases. 3rd IEEE Conf. on Data Engineering, Los Angeles, (March 1987), 246-250.

[Wie89] Wiederhold, G. The architecture of Future Information Systems. Draft. Stanford University, (Jan. 1989), 17.

[Wol89] Wolski, A. LINDA : A System for Loosely Integrated Databases. 5-th IEEE Conf. on Data Engineering, Los Angeles (Feb 1989).