Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

An Organic Database System

Martin L. Kersten, Arno P.J.M. Siebes

Information Systems (INS)

INS-R9905 April 30, 1999

Report INS-R9905
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

An Organic Database System

Martin L. Kersten, Arno P.J.M. Siebes
{mk,arno}@cwi.nl

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT
The pervasive penetration of database technology may suggest that we have reached the end of the database research era. The contrary is true. Emerging technology, in hardware, software, and connectivity, brings a wealth of opportunities to push technology to a new level of maturity. Furthermore, ground breaking results are obtained in Quantum- and DNA-computing using nature as inspiration for its computational models. This paper provides a vision on a new brand of database architectures, i.e. an Organic Database System where a large collection of connected, autonomous data cells implement a semantic meaningful store/recall information system. It explores the analogy of a biological complex to charter the contours of this research vision. A concrete computational model is defined and illustrated by examples as a step into this direction.

1991 Mathematics Subject Classification: 68P20,68P15,68N99
1991 Computing Reviews Classification System: Database Logical Design(H.2.1), Database Systems (H.2.4)
Keywords and Phrases: database architectures, distributed information systems
Note: Work carried out under INS-1 Speculative Database Research

1. Introduction

The innovation thrust of current database research comes from attempts to deploy its technology in non-trivial application areas. Enhancements proposed to the core functionality are primarily triggered by the specific needs encountered, e.g. GIS, multi-media, and data mining.

The purpose of this paper is to confront the community with a vision on an alternative track for database architecture research. One grown over a decade as a temporary escape from our contract-research work, which dictated most of our agenda. As such, the vision presented is by no means complete, nor explored in all its depths, let alone be implemented in a eye-catching demonstrator. It is, however, a significant step forward from an earlier version, presented at the RIDE 97 workshop on research issues in Databases and published in [Ker97].

The premises is that database technology has contributed significantly to society over several decades, BUT it is also time to challenge its key assumptions. A few issues considered to be dogmatic and a bottleneck for progress are:

- A database is larger then the main memory of the computer on which the DBMS runs and a great deal of effort should be devoted to efficient management of crossing the chasms between disk and memory.
- A DBMS should adhere to a standard datamodel, whether it be relational, an object-relational, object-oriented, and leave functional and deductive models as a playground for theoretical research.
- A DBMS provides quick response to any (unrealistically complex) query, optimizing resource usage wherever possible without concern on the effect of concurrent users.
- A DBMS should support concurrent access by multiple users at the smallest granularity level (record) and reconcile the different perspectives on the database contents transparently.

- A DBMS provides a transaction models based on the ACID principles, or a semantically enriched version, regardless its primary domain of application.

This list is by no means complete, but merely indicates the delineation of research activities in the database community. A prototype DBMS ignoring these points is not taken seriously in the research realm. Albeit, in each of the assumptions reality is threatening, e.g. the web challenges the rigid data models, the transaction models, and replication management.

The research agenda derived at the Asilomar workshop [1] rightfully acknowledge that in the near future all but the largest relational tables will be memory resident, calling for a complete overhaul of the current data structures, algorithms and system architecture. A grand challenge is subsequently defined for the database community: *The information utility*: make it easy for everyone to store, organize, access and analyse the majority of human information online.

The key question then boils down to "Is the current architectural conception of database technology a sufficient basis to meet this challenge? " Our preliminary answer to this question is negative. The threats to the database dogmas are evidence of their failure. Instead we need a more unorthodox approach to break our historical bonds. A broader perspective on computer science research may be of help here.

Recent major fundamental advances in computer science have found their inspiration in nature. DNA computing uses biochemistry to implement massive parallel computation and the engineering of DNA computing device seems tractable [PRS98]. Quantum computing uses the physics of light to design a new computational model. Hurdles to be taken here include non-destructive observation of the result of a computation. Theoretically both quantum- and DNA-computation have been shown to crack hard problems in cryptography. Further back into the history computing, we find the notion of self-reproducing automata [Neu66]. A study severally hindered by the state-of-the-art in computer architecture, but nevertheless an intriguing concept. A more mundane use of nature as a stimulus for novel programming paradigms have let to such broad fields as evolutionary computing and neural nets. A large community deploys these concepts to realize new kinds of applications, e.g. adaptive, intelligent, and even socially acceptable agents [HS98].

With these examples in mind, it is worth considering how nature can inspire us in the design of a new database management paradigm. The remainder of this paper charters the contours of such an Organic Database System, i.e. a large collection of connected, autonomous data cells that implement a semantic meaningful store/recall information system.

In a nutshell, the architecture is centered around the concept of a *data cell*, characterised by three components: an interface, a cell body and a nucleus. The cell interface is a semi-permeable membrane taking two forms; RECEPTORS, where objects in the cell's environment may enter the cell; and EMITTERS, which enable objects in the cell's interior to migrate to the outer world. The cell's body is a memory structure for the tree-structured objects received. It is a persistent store organised by object entrance/creation time. The nucleus consists of genetic code strings interpreted under triggering events, i.e. objects stored in the cell's body.

As such, a data cell models a physical container capable of managing a small database, limited in capacity, without a fixed foothold, and equipped with behavioral knowledge, described by RECALL-FORGET-KEEP genes, which replace the role of procedural methods.

The cells live in a resource rich environment, which enable them to migrate or to clone as soon as the physical boundaries are met. Communication amongst data cells is modeled after arteries, neurons, and membrane sharing found in nature. They model different modalities of communication. The Internet is assumed as the underlying communication network, where cells are addressable with an URN.[2] To survive in this dynamic world, a cell may decide to seed a copy of its state. It is resurrected upon request to recover from a physical disaster.

Querying the organic database is a non-atomic process. A query is mapped to a membrane modi-

[1] http://www.acm.org/sigmod/record/issues/9812/asilomar.html
[2] http://www.acl.lanl.gov/URN/

fication that allows answers to pass to the communication interchange, where they can be picked up by the issuer. Since cells may be temporary dormant or inaccessible, the issuer should be prepared to wait for all cells to respond or be satisfied with partial answers. The net effect is that querying becomes probabilistic, much like searching the Web. One never knows for sure if all information has been obtained.

The remainder of this paper is organized as follows. Section 2 introduces the data cell, its internal architecture, and a notation for reasoning. Section 3 introduces the modalities of communication. Section 4 addresses the life cycle of an individual cell, its sensors, its cloning, and seeds to survive disasters. Section 5 presents a small application with mockup traces to illustrate the projected behavior. Section 6 boils down to a summary, and raises some fundamental research issues to be considered next.

2. Data Cell Overview

In this section we introduce the notion of a data cell, the basic building block of an Organic Database System. We take an inward exploration, starting with the cell's membrane, followed by its memory structure, and to finish with the behavior described by its nucleus.

2.1 The data cell and its membrane

A data cell is a physically bounded resource to store and recall persistent information. Physically boundedness is interpreted as anything from a simple smart-card in a mobile phone, up to large multi-processor SMP machine. In our search for a new architecture we favor the former, because it challenges us to go for minimal and razor-blade components. An SMP context merely leads to challenging engineering issues related to scale.

The data cell and its membrane are defined as follows:

Def.1 A data cell type $D =< M, N, T >$ named D, consists of a set of receptor- and emitter-membranes $M = \{M_i\}$, a set of genetic code strings in the nucleus $N = \{N_j\}$, and $T = \{T_k\}$ a collection of data types.

Def.2 The instances of a data cell type D are denoted by the type Cid, i.e. globally unique, life-time tags.

Def.3 Object structures are defined by a context-free grammar $G =< N, V, P, D >$ with non-terminals N, terminal variables V, their productions P.

Def.4 A RECEPTOR membrane is defined by the structure: $Name$ RECEPTOR W WHERE $P(B)$ where $Name$ is an optional membrane tag, W a derivation tree for a term in L(G) starting at D, and B the binding table for variables in W.

Def.5 An EMITTER membrane is defined by the structure: $Name$ EMITTER W WHERE $P(B)$ where $Name$ is an optional membrane tag, W a derivation tree for a term in L(G), and B the binding table for variables in W.

The membrane definitions are based on our conjuncture that most objects for information exchange can be described formally, and exhaustively, with a context-free grammar.[3] This grammar provides a structured name space to access and to reason about the components. The degrees of freedom lie in the production rules, i.e. the type constructors, and the lexical tokens, i.e. the data types. This relationship between structure and values is factored out in the binding table B. A parse tree for an object then contains bound variables as leafs.

The last component of a membrane is a predicate over the object components, represented by the (dynamically typed) variables V. The predicate is safe when all its variables are bound. Otherwise the predicate fails. The predicate language relies on operators defined for the data types T. For all practical purposes considered in this paper, we assume T to include the standard set of basic types available in the programming environment. Furthermore, T includes N, the grammar non-terminals and Cid, the cell identities.

[3]Notational convention: identifiers starting with a lower-case character act as cell names, their components, and as object structure tags. Those starting with an upper-case character are used as variables.

Def.6 A membrane M_i for data cell D accepts (emits) objects from (to) its environment if it satisfies both the structure implied by the grammar G, the values T, and the predicate $P(B)$ holds.

The parser derived from the RECEPTOR membrane grammar looks for external object structures tagged by the cell type name, i.e. the grammar's start symbol. The object structures are stored in the cell's body. The EMITTER looks for qualifying object structures in the cell body. Their top level name denotes the target cell for the message.

Receptors and emitters map to autonomous threads. They may inspect objects concurrently in their environment, but only one may finalise the transaction (passage of a cell's membrane). Objects rejected by all RECEPTORS are left to the responsibility of the communication environment. Objects failing the EMITTER membrane remain in the cell body.

A full fledged implementation of the data cell may exploit standard notational conventions, such as XML or ODL object structures. The sole requirement for the object description language is that a message can be mapped onto the grammar and binding table of the membrane, i.e. the parse is unambiguous irrespective the lexical convention. A concrete syntax for cell identities *Cid* could be an URN.

Example. As a running example we consider a toy database of colored marbles. The data cell defined below looks in its environment for red marbles. All encountered are catched and stored in the cell's body as `marble`("red"). The first emitter may pick up the marbles again and thrown them back into the environment. The second emitter looks for marbles changed (by magic) to those with a primary and secondary color. They are sent to cells interested in multi-coloured marbles. The last part illustrates initialization with a few marbles.

```
CELL marble;
    RECEPTOR K WHERE K = "red";
    EMITTER marble(K) ;
    EMITTER marble(Primary,Secondary)
        WHERE Secondary = "orange";
    marble("orange");
    marble("green");
    marble("yellow","orange");
END marble;
```

2.2 Structure unification terms

The organic database system exploits the equivalent of Watson-Crick complementary feature provided 'for free' by the nature. This feature stipulates the programming power in DNA-strands, where bases opposite each other are complementary. During the construction of the double helix strands, genes are *unified* with the nucleotides to find matches.

The equivalent notion exploited here is to unify object terms in predicates with the parse trees of the object structures received. Unification is supported by the operator ':', i.e. the term $X : Y$ succeeds if the operands can be unified. The terms considered are classified into (un)ordered- and (un)tagged- object terms.

- *ordered* ($X_0, \cdots, X_j$), which exhausts a single object component list structure.
- *unordered* { $X_i, \cdots, X_j$ }, where all elements mentioned denote path expressions binding different components in a single object structure.
- *prefixed*, `cntxt`$(X_0, \cdots, X_j)$ and `cntxt`{ $X_i, \cdots, X_j$ } are called prefixed object terms, all components mentioned belong to a single object structure reachable through the path expression `cntxt`.

Example. Consider the term `marble(primary(`P`), secondary(`S`))`, which unifies with any variable Z. Subsequently Z can also be unified with `marble`(X,Y) and $M(X, Y)$ where M binds with `marble`.

The unordered unification $Z : \{\texttt{marble.primary}(X)\}$ and $Z : \{\texttt{primary}(X)\}$ hold, because the paths exist in the structure referenced by Z. The prefixed terms $Z : marble(primary, secondary)$ and $Z : marble\{primary, secondary\}$ hold, while $Z : \{\texttt{marble}.P, \texttt{marble}.Z, \texttt{marble.secondary}\}$ fails, because the three arguments can not be bound to different object components. Finally, we permit unification with a type name to denote membership, e.g. "red": `string` also holds. With this notational convention, predicates over the hierarchical object base becomes condense and easy to interpret.

2.3 The cell's body

The cell's body is a persistent memory structure, where objects passing the membrane are kept. Its organization affects the subsequent computational model, both internally and externally. One extreme is to consider memory as a set of tree structured objects, freely floating within the cell's body. The effect is that all sequential behavior calls for 'sorting', or the cell behavior becomes purely probabilistic. Given that nature also processes cell DNA strands in sequential fashion, we choose for a time-organized sequence.

The memory sequence comes with two maintenance operations: KEEP and FORGET. A KEEP V operation adds the object V to the end of this sequence in an atomary step, while FORGET V 'zaps' the (bound) object from the memory sequence, leaving no traces behind. Information in the memory sequence can be located with a RECALL operation followed by an ordered list of object terms. Its semantics is to traverses the memory sequence in reverse direction, i.e. it unifies terms to the latest objects entered. Moreover, no two terms in the recall list unify to the same object (component).

Example. The table below illustrates a memory sequence. The right part illustrates the successive term unifications that result from the RECALL `marble(X),marble(Y,Z)`. Note, the two red marbles are distinct objects, although their structure and value are identical.

marble	Memory sequence	X, (Y,Z)
0	marble("red")	0 ,1
1	marble("red","orange")	2, 1
2	marble("yellow")	3,1
3	marble("red")	

2.4 The cell's nucleus

The nucleus of a cell contains a set of chromosomes, gene strands, that defined its behavior. A gene strand consists of RECALL - FORGET-KEEP statements. Each gene inspects and changes the memory sequence under all possible variable bindings.

Def.7 A gene G is described by the structure:

G RECALL L WHERE $P(L)$
  FORGET Fl WHERE $P(L \cup Fl)$
  KEEP Sl WHERE $P(L \cup Fl \cup Sl)$;

where G is an optional gene tag, L, Fl, and Sl term lists to locate objects, and $P(\alpha)$ a clause over the variables in the term lists indicated.

The interpretation of a gene is that each completed binding leads to an atomary action against the memory sequence. Some objects bound are prepared for removal, and new object structures are prepared to inclusion. These changes take immediate effect for each term binding encountered.

Def.8 A chromosome is a structure $G =$ RECALL $Cl\{G_0; \cdots ; G_k\}$ where Cl is a term list and G_i is either a gene or a chromosome sequence.

The scope of variables introduced in the chromosome RECALL list is defined by the corresponding gene sequence. For simplicity we assume no redefinition of variables.

Def.9 A chromosome G is independently activated for each bindings of its memory recall list Cl.

The chromosome describes a hierarchical sequence of behavioral actions. The RECALL is an implicit loop through memory and the qualified update statements are guarded commands. Conceptually, each time an object appears in the cell's body it will arbitrarily activate a chromosome interested

in the object. All valid bindings are explored before the chromosome ceases activity. Since binding works its way back into object history, and KEEPs are always at the head of the sequence, this process will eventually terminate. A limited set of additional functions controls the life-cycle of a cell. This includes ENABLE/DISABLE of cell components, WAKEUP peer cells, going to HIBERNATE, RUNning linked in routines, and CLONEing itself.

Example. The intended behavior of the cell nucleus is illustrated using our marble toy database. Each time the membrane stores an object `marble`("red"), the nucleus is inspected for a qualifying chromosome, i.e one whose first element in the RECALL term list unifies with the object. Once detected, a process thread interprets the chromosome, consuming the object and creating a new object for emission later on.[4]

```
CELL marble;
    RECEPTOR K WHERE K = "red";
    EMITTER marble(K) WHERE K ≠ "red" ;
NUCLEUS
    RECALL marble(Msg)
        FORGET IT
        KEEP marble("green");
END marble;
```

The probabilistic behavior of chromosome selection and their inter relationships are illustrated below. In the next fragment one chromosome arbitrarily transforms a red marble into either green or orange. With each color change we also remove any trace of the red marble received. Note that the probabilistic behavior envisioned may also lead to emission of red marbles, before they are inspected by any of the chromosomes. They may then end-up in cells capable to react.

```
CELL marble;
    RECEPTOR K WHERE K = "red";
    EMITTER marble(K) ;
NUCLEUS
    RECALL marble("red")
        FORGET IT
        KEEP marble("green");
    RECALL marble("red")
        FORGET IT
        KEEP marble("orange");
END marble;
```

The second fragment replicates a red marble into both red and green by being bound with each chromosome once. Moreover, it accumulates the red marbles in its memory, because they are never forgotten.

```
CELL marble;
    RECEPTOR K WHERE K = "red";
    EMITTER marble(K) WHERE K ≠ "red";
NUCLEUS
    RECALL marble("red")
        KEEP marble("green");
    RECALL marble("red")
        KEEP marble("orange");
END marble;
```

[4]The keyword IT stands for all objects bound in the memory recall list.

To get rid of the red marbles too, we have to encode state information in the data cell. A possible solution using tagged intermediate results is shown below. The tag is attached to each object indicated by the KEEP to indicate the chromosome responsible for its creation.

```
CELL marble;
    RECEPTOR K WHERE K = "red";
    EMITTER marble(K) ;
NUCLEUS
    RECALL marble("red")
        KEEP m1("green");
    RECALL marble("red")
        KEEP m2("orange");
    RECALL m1(A), m2(B), marble("red")
        FORGET IT;
        KEEP marble(A), marble(B);
END marble;
```

3. The Communication System

Data cells in isolation are of limited use. A communication structure gives the Organic database system access to its sensory components and circumvents the physical boundaries imposed by hardware. This section describes the analogue of biological communication schemes in the context of our Organic Database System.

3.1 Artery system

Nature has found an efficient solution for communication in the form of arteries, where the transport medium need not worry about the message content. It merely passes objects around and leaves it to the autonomous cells attached to the artery to filter out objects of interest.

We use this scheme for the organic database as the backbone communication infrastructure. It consists of artery segments, which are effectively containers for a limited number of message objects floating through the system. Furthermore, each segment shares a membrane with (a limited number of) data cells, looking for messages of interest passing by. These cells get access to the messages in a probabilistic manner. Furthermore, the segment may be linked with sensors to the outside world, giving it eyes and ears to communicate with the user.

Example. The artery segment `s1` in Figure 1 can be modeled as a data cell shown below. It accepts any message from `s0`. The chromosomes pick objects at random and pass them onwards to the next segment `s2` or `s3`.

```
CELL artery
    RECEPTOR s0(Msg);
    EMITTER artery(Dest(Msg));
NUCLEUS
    RECALL  s1(Msg)
        FORGET IT
        EMITTER artery(s3(Msg));
    RECALL  s1(Msg)
        FORGET IT
        EMITTER artery(s2(Msg));
END artery;
```

The artery system provides a natural communication scheme, but also possesses some dangers. First, the artery system may become polluted with messages of no interest to any cell. Second, the probabilistic flow does not guarantee that a message will reach a destination cell in acceptable

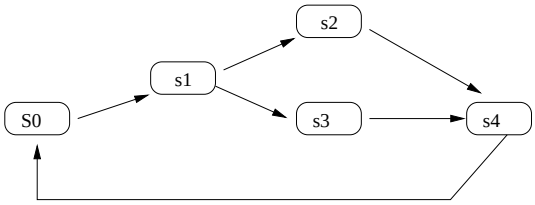

Figure 1: Artery flow expansion

time. Although this reflects real-life on the Internet, it may be unacceptable in a confined application environment. The solution to consider then is to introduce many cells on the artery system, such that the probability steeply increases. Alternatively, a multi-level artery system can be designed through which messages quickly reach their intended destination. For example, nature often uses a nerve system to sent simple information around quickly. This includes intermediate control centers to handle local issues and shortcuts.

Example. Pollution of an artery segment with unwanted messages can be controlled by tagging them with an age component. A single cell will remove them as waste when they get too old. The artery cell below is charged with this functionality.

```
CELL message
    RECEPTOR (M, age(C));
    EMITTER message(X,Y) ;
NUCLEUS
    RECALL  (M, age(C) )
        WHERE C<=1000
        FORGET IT
        KEEP message(M, age(N)) WHERE N is C+1;
    RECALL  (M, age(C) )
        WHERE C>1000
        FORGET IT;
END message;
```

3.2 Neurons

Every data cell carries an unique identifier. Knowing this identifier permits direct addressing of a target cell using a dedicated transport scheme. It merely has to be constructed. Nature's realisation for this can be found in neurons. It is a one-to-one communication channel, orthogonal to the artery system. They have to be 'learned' and they involve much less overhead in terms of communication and analysis.

A neuron can only fire when the target cell is alive, leaving an object at the target cell's membrane for direct inclusion. In this sense, neurons communication can be seen as a kind of synchronous communication. This makes them part of the processing thread(s) of the nucleus, where they block progress until the message is delivered.

Example. The fragment below illustrates how a marble cell handles a query of a client. The client issues the request `getAll`(SELF) to pass its identity to the marble cell. It expects copies of the objects to arrive at its membrane in return. The marble catches the request with the first chromosome, and activates the neuron stream of answers. It also illustrates a complex chromosome with sequential behavior.

```
CELL marble
```

```
    RECEPTOR getAll(Mid);
NUCLEUS
    RECALL  getAll(Msg) {
        FORGET IT;
        RECALL client(Mid), Object
            NEURON Mid(Object);
        }
END marble;
```

3.3 Membrane sharing

The third communication scheme between cells is based on sharing a MEMBRANE definition, followed by a term list argument. Such a membrane describes externally stored, but accessible object collections. That is, two cells `marble#0` and `marble#1` sharing a membrane permit inspection and updates of one-anothers cell body in a chromosome. The objects satisfying the membrane freely move between the cells.[5] This way it becomes easy to construct data-distributed applications.

Example. The fragment below illustrates two instances of a marble cells sharing the term membrane. This makes the marbles stored directly accessible to the other cell. When cell `marble#1` finds an orange marble, it will remove any green marble in cell `marble#0` and `marble#1`. Conversely, `marble#0` looks up any red marble in both bodies and transform it to green. As such, the cell are functionally specialized.

```
CELL marble#0;
    RECEPTOR K WHERE K ≠ "orange";
    MEMBRANE marble(X);
    marble("green"); marble("red");
NUCLEUS
    RECALL marble("red") KEEP marble("green");
END marble;

CELL marble#1;
    RECEPTOR K WHERE K = "orange";
    MEMBRANE marble(X);
NUCLEUS
    RECALL marble("orange"), marble(Z)
        WHERE K = "green"
        FORGET marble(K);
END marble;
```

4. The Life Cycle

The textual definitions given for the data cells are their 'seed' state. They can be resurrected from this state by an external entity, which is typically a organic database system kernel implementation or a cell using a WAKEUP call. Once active, it can CLONE itself, and return to HIBERNATE state as part of its nucleus behavior. These issues are described below.

4.1 Cloning a data cell

A data cell may CLONE itself to form a data cell tissue, a collection of cells with identical behavior. This process is triggered by a nucleus action and consists of two phases. In the first phase, all activity is stopped as if the cell M goes into hibernation state. Then a textual 'seed' copy C is created, which contains half of the memory sequence, a MEMBRANE definition, and the object `parent`(M). This phase ends with forgetting the objects moved to the clone and placing the object `child`(C) and the

[5] Subject to a proper semantics of the memory sequence.

MEMBRANE definition in the body of M. Following, in phase two, cell M becomes receptive to external requests again when the object `child` is forgotten. Cell C follows the normal awakening sequence, where it will react to the *parent* object before it accepts any further request.

A major difference between cloning and the creation of a cell is that a clone is connected to its heritage via membrane sharing. This sharing allows the cell tissue to act as one cell as far as data storing and retrieving is concerned. To an outsider it is immaterial which cell in a tissue acts upon his request as long as it is acted upon.

At the moment a cell is cloned -or created- the artery system has to be adjusted as well. The identity of the new cell should be announced to this communication channel.

4.2 Hibernation and wakeup

Hibernation is a multi-step procedure. First, the cell's receptors are deactivated, the chromosomes are instructed to stop as soon as possible in a recoverable state, the emitters finish sending all qualifying objects. They also stop when the environment does not accept the objects emitted anymore. Finally, the cell status is saved to disk.

Example. The marble cells goes into hibernation after receiving a "blue" marble.

```
CELL marble
    RECEPTOR marble("blue");
NUCLEUS
    RECALL marble("blue")
        FORGET IT
        HIBERNATE;
END example;
```

A dormant cell can be awakened by any cell using a WAKEUP call passing the cell's identity. This typically takes place in an artery segment, triggered by the cell name in a message header. If an artery segment runs out of resources, it may decide in a probabilistic manner what cell to ask for hibernation or to migrate a cell under its control to another segment.

An awakened cell starts with RECEPTOR and EMITTER elements in passive mode first. They should be activated by a chromosome. The triggering event is existence of the object 'resurrected' in the cell's body. This unifies with the corresponding initialization chromosome. The default -shown below- looks up all (still passive) membrane structures and activates each.

```
NUCLEUS RECALL resurrected {
    RECALL E:EMITTER { ACTIVATE E;}
    RECALL R:RECEPTOR { ACTIVATE R;}
    FORGET resurrected;
    }
```

5. Application Challenges

In this section we illustrate the Organic Database System using a distributed `phone` book, one whose data cells may indeed live in our digital organizer, our PC, and mobile phone concurrently. As such it is close to what one would expect from a store/recall information system. We start with a sensory interface, the eyes and ears of the system. Following we give a concrete definition of the phone book, one that will not (!) immediately work, but which highlights the issues to be dealt with. Finally, we indicate routes to implement an organic database system.

5.1 Sensors

The data cells introduced so far were blind actors. They communicate amongst one-another using the biological inspired schemes. However, at least one cell should provide a bridge to the real-world, where we observe and control the behavior of an organic database.

This calls for the equivalent of sensors, the eyes and ears of the system. Since the functionality of sensors are tightly coupled with the environment where they operate, it has to rely on linked-in libraries. The minimal set to be considered for a first implementation are a direct link to the `stdio` library and XML for web-based interaction.

A sensor cell has an event loop triggered both by the external interface and the messages from the cells. The latter are screened for type correctness. Subsequently, they may be picked up by a chromosome to be executed. This essentially makes a sensor cell a wrapper around a user-supplied interface library.

Example. The ascii sensor below assumes an io-library, which interacts with the user through an text-based interface.

```
CELL ascii USE stdio;
    RECEPTOR print(Msg:string);
    RECEPTOR printf(Format:string,Msg:string);
NUCLEUS
    RECALL Action RUN stdio.Action;
END ascii;
```

5.2 A phone book

Figure 2 illustrates the starting position of the phone cell. This definition is no more complex than a class definition in an object-oriented paradigm, or an SQL-3 table definition. Each time an object passes the membrane it is added to the persistent store, as is to be expected from a database system. Using the textual interface we might add some persons.

```
> phone.person(name("Smith"),tel(808717));
> phone.person(name("Jones"),tel(828503));
> phone.person("Jones",tel(808717));
```

The structure for Jones does not match the receptor, leaving it in the artery. A waste recovery cell (See Section 3.1) can be used to get rid of these messages. Alternatively, we could accept a broader class of person structures and emit an error message where appropriate. The necessary additions become:

```
RECEPTOR person;
EMITTER error(X);
RECALL P:person
    WHERE NOT(P:person(name(N),tel(T)))
        KEEP error(P);
```

Lookup of Smith's telephone number is straight forward requested by:

```
> phone.lookup("Smith);
```

However, the user does not know when the answer will arrive, because, due to cloning, the actual cell containing this information may be temporarily inaccessible. This is generally the case with interrogation of an organic structure. Getting an answer to the question "Is Smith not part of the phone book", can only be answered from the contextual knowledge that all cells are active and have dealt with this question.

The cell also contains a chromosome to clone itself when it accumulates too many telephone entries. Its controlling predicate is an aggregate over the memory sequence. However, such constructs require extreme care, because membrane-based sharing implies that we always count **all** elements in the data tissue. This leads to a cascade of clones after receiving 10 persons. A way out of this dilemma is to consider non-sharing clones or quantified bindings, i.e.

```
CELL phone;
    RECEPTOR person(name(N),tel(I))
        WHERE N:string AND I:integer;
    RECEPTOR lookup(name(N))
        WHERE N:string;
    RECEPTOR delete(N));
    EMITTER answer(Msg);
NUCLEUS
    RECALL lookup(N) {
        RECALL person(name(Nme),T)
            WHERE Nme == N {
                FORGET  lookup(N);
                KEEP  answer(N,T));}
    }
    RECALL delete(N) {
        RECALL person(name(Nme),T)
            WHERE Nme == N FORGET IT;
        FORGET delete(N);
    }
    RECALL count(P:person) > 10 CLONE;
END phone;
```

Figure 2: The phone book

```
    RECALL count(ALL LOCAL P:person)>10 CLONE;
```

Being able to inject a receptor, emitter, or chromosome into a cell modifies its behavior. This is particularly useful if we want to extract information, i.e. query its content. A 'virus' cell penetration of the phone book might be a road to explore. It may take the following form:

```
CELL virus;
    EMITTER steal(X);
NUCLEUS
    RECALL resurrected
        ACTIVATE steal;
    RECALL Y KEEP steal(Y);
END virus;
```

Once we are able to bind this virus with an object passing the phone book membrane (possibly as a Trojan horse), it awakes from its hibernated state. Both symbiotic and harmful viruses are easy to design.

5.3 Implementation strategies

The organic database system outlined does not require a start from scratch. Many ingredients for its realization are readily available.

The history for the cell architecture can be traced back as far as the Von Neumann cellular automata, a dream where computers conquer free space to grow and solve intricate problems [Neu66]. The abundance of literature on cellular automata in the 60s provide further hints to formalize parts of the data cell semantics[Wolf82]. The work on associative memories [Ozk86, Sue88] in the 70s can be regarded as preliminary steps to improve the cell's memory.

The declarative model underlying the interrogation of the cell's memory sequence is a natural extension to SQL- and logic-based systems. Modern prototype database engines to be considered are Lore [Lore97] and Monet[BQK96, BKK96, BWK98]. They provide a lean implementation to start from. Furthermore, Java-beans technology may be a pivot in realisation of cells with a small footprint.

Likewise, the problems posed by cloning find their analogy in distributed belief systems. Recent developments in agent technology, especially the libraries being developed for agent-based applications on the web, may provide the seeds to quickly build a prototype organic database system.[HS98]

The data sharing that results from the membrane replication, may use techniques from distributed computation models, such as explored in Linda [Gel85]. The temporal aspects of the memory sequence can be borrowed from [Snod94].

6. Conclusion

This paper is written at a time that deployment of our Monet system [BQK96, BKK96, BWK98] is in full swing and it capitalizes experience gained in producing several other full-fledged systems [WKSP86, PRISMA92]. It aligns with trends of commercial DBMS providers, who have already recognized the limited growth in their core products and embrace complementary solutions, e.g. Sybase with IQ, Computer Associates with Jasmine, Oracle with Express, and Tandem with its Data Mining System. It is safe to predict that within the next decade they provide a DBMS product configured as an assemblage of specialized servers.

Nevertheless, this progress in technology lacks a quantum leap. It is largely an engineering activity, continuously backlagging the evolution in hardware, software, and application needs. To break out, the vision developed in this paper provides an innovative computational model, data model, and architecture for database processing.

We consider this paper a success, if the reader has raised questions about the limitations of the organic database approach, thought of refinements, or envisioned a concrete realization. Moreover, the biological metaphor may have to be extended into other fruitful directions or being corrected as a result of our limited knowledge on the biological mechanics.

The research road ahead is thus marked with many fundamental questions calling for indepth studies. Amongst these, the most pregnant are to describe cloning as reflexive behavior, to use symbiotic behavior, and to better understand the implications for large-scale application development. At an architectural level, embedding data cells in hardware ranging from smart-cards up to super-computers with their wildly differing communication infrastructure stresses the need for strong interface definitions. Perhaps a Jini from a bottle talking XML comes to rescue.

References

[BQK96] P. Boncz, W. Quak, and M. Kersten. Monet and its geographical extensions. In *Proc. EDBT'96*, Volumne 1057 *Lecture Notes in Computer Science*, Avignon (France), March 1996, pp. 147-166.

[BKK96] P. Boncz, F. Kwakkel, M.L. Kersten. High performance support for O-O traversal in Monet. In *proc. BNCOD'96*, Edingburgh (UK), July 1996.

[BWK98] P. Boncz, A.N. Wilschut, and M.L. Kersten, Flattening an Object Algebra to Provide Formance. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, Orlando, Fl, USA, 1998.

[Gel85] D. Gelertner. Generative Communication in Linda. In *ACM Trans. Prog. Lang. and Systems*, 7 (1), 1985.

[HS98] M.N. Hunhs, M.P. Singh. *Readings in Agents*. Morgan-Kaufmann,1998, ISBN 1-55860-495-2.

[PRISMA92] P.M.G. Apers, C.A. van den Berg, J. Flokstra, P.W.P.J. Grefen, M.L. Kersten, and A.N. Wilschut. PRISMA/DB: A Parallel, Main-memory Relational DBMS, In *IEEE KDE, special issue on Main-Memory DBMS*. 4(6):541-554, December 1992.

[Ker97] M.L. Kersten. A Cellular Database System for the 21th Century. In *Proceedings ARTDB'97*, Springer-Verlag, Sep. 1997, pp..

[Lore97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. In *SIGMOD Record*, 26(3):54-66,September 1997.

[Sue88] S. Sue. *Database Computers, Principles, Architectures & Techniques*, Mcgraw-Hill 1988, ISBN 0-07-062295-7.

[Neu66] J. von Neumann. The General and Logical Theory of Automata. In *J. von Neumann. "Collected Works"* (ed. A.H. Taub), Vol. 5, p.288; J. von Neumann, "Theory of Self-Reproducing Automata", (ed. A.W. Burks), Univ. of Illinois Press(1966); ed. A.W. Burks, "Essays on Cellular Automata", Univ. of Illinois Press(1970).

[PRS98] G. Paun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*. Springer-Verlag, 1998, ISBN 3-540-64196

[Ozk86] E. Oskarahan. *Database Machines and Database Management*. Prentice Hall, 1986, ISBN 0-13-196031-8.

[Wolf82] http://www.wolfram.com/s.wolfram/articles/82-cellular/index.html

[Snod94] R.T. Snodgrass et.al *The Tsql2 Temporal Query Language*. Kluwer International Series in Engineering and Computer Science; No. 330.

[WKSP86] A.I. Wasserman, M.L. Kersten, D. Shewmake, P. Pircher. Designing information systems within the User Software Engineering environment. In *IEEE Transactions on Software Engineering*, 12(2):326-345, Feb. 1986.