Regardless of the size of your computer system it is a strategic tool. We are committed to helping you use it most effectively. All it needs is a little imagination.

UNIDATA
Telephone (012) 45-1152

THERE'S A GIANT IN EVERY ONE OF US
The Department of Computer Science at Pretoria University is inviting applications for two annually awarded bursaries for MSc study. These bursaries are aimed at outstanding students who have a Computer Science Honours degree (or equivalent qualification). As far as possible, one such bursary will be awarded to a South African student, while the other will be awarded to a student from another African state. The hope is to thereby promote interaction between South African computer scientists and computer scientists elsewhere in Africa.

The bursaries are funded by the Department's contract research work and will amount to a minimum of R20000 per year. They are intended to cover costs for tuition, housing, transport, books, and an adequate allowance for other living expenses. Successful candidates will be required to work for 20 hours per week as part of the Computer Science Department's contract research team. The work assigned to each candidate will be aimed at providing him with practical experience in his field of interest.

Study fields for which bursaries will be considered include:
- Computer Networking
- Distributed Systems
- Software Engineering
- Artificial Intelligence
- Intelligent Computer Aided Instruction
- Machine Aided Translation

Students should aim to complete the dissertation that is required for an MSc within two years of study (i.e. one year of full time study). The dissertation should preferably be written in English but it may also be written in Afrikaans.

The continuation of the bursary and adjustments to the allowance will be decided upon annually and will depend on progress. It is a condition of the bursary that, after completing his/her studies, the student should return to his/her country of origin for the same number of years for which the bursary was held.

Applicants may request application forms from:

MSc. Bursaries
The Computer Science Department
University of Pretoria
Hillcrest
Pretoria
0083
South Africa
Editor's Notes

It is with sincere gratitude that SACJ takes leave of Dr Peter Lay who, until recently, was the assistant editor dealing with Information Systems. He has left academia for what sounds like a more gentle lifestyle. (He has gone farming!) Under Peter’s stewardship the number of high-quality IS papers in SACJ grew steadily. In general, IS papers tend to be accessible and relevant to a wide spectrum of computer professionals, and the quality of IS papers that have been appearing in SACJ has significantly contributed to the increased interest being shown in the journal by the local computer industry.

If this growth in interest is to be sustained, it is urgent and important to find a suitable replacement assistant editor. The ideal candidate should not only be respected as an academic by his peers, but should also be disposed to enthusiastically promote SACJ in the private sector. Since a shortlist of candidates is currently being compiled, I would like to issue a general appeal for names that might be included on it. Please contact me urgently if you would like to be considered for the job, or if you would like to nominate someone that you consider to be particularly suitable.

My three year term of office as editor expires in October. I have always considered it a great privilege to hold this position, and as a result, I felt honoured when the SAICS executive committee requested that I stay on for a further term. Nevertheless, I initially declined the request on the grounds that the time-demands of the job were significantly eroding my ability to fulfil other duties. Particularly demanding has been the task of seeing to the typesetting of the various contributions - either by doing it myself, or by ensuring that it is adequately done by someone else. Recently, however, Prof G de V Smit (Riel Smit) at UCT has offered to assume the role of production editor. This generous offer so much changes the complexion of what is being asked of me that I am now both willing and honoured to continue as editor for another term. I am very grateful to Riel for his offer and I look forward to working with him. In future, authors whose papers have been accepted for publication will be asked to liaise directly with him regarding the precise form in which the final contribution should be submitted.

The next issue of SACJ will consist largely of a selection of papers that were presented at the 6th South African Computer symposium. The selection will be based on comments from the referees who, at the time, were asked to adjudicate the papers in terms of their appropriateness for both the conference as well as for SACJ publication. Papers which, in the opinion of one or more referees, required major revision will have to be resubmitted to SACJ for refereeing purposes. Authors will soon be contacted in this regard.

At the time of writing, the updated list of "approved" publications for the first half of 1991 had not yet been released by the relevant authorities. For the sake of past, present and future contributors I sincerely hope that SACJ will be on the list when it eventually comes out. However, I have become increasingly aware that there is a real danger of laying too much store on papers published in so-called approved journals as a basis for evaluating and rewarding research. I hope to expand more fully on this theme in a future edition of SACJ. Keep watching this space!

Derrick Kourie
Editor

This SACJ issue is sponsored by a generous donation from UNIDATA
Concept Network Framework for a Multi-paradigm Knowledge Base

J Kambanis
Department of Computer Science and Information Systems, UNISA, P.O.Box 392, Pretoria, 0001

Abstract

This paper describes a conceptual framework and an implementation medium for constructing multi-paradigm knowledge bases. The idea of a concept network is used as a basis for storing knowledge in different knowledge representation formalisms. Concepts from one formalism can refer to concepts from another, and the same knowledge can have different meanings in different contexts. It is based on a three layer system: the bottom layer is an associative network with no semantics, the middle layer is a semantic layer where the semantics of different formalisms are written using primitive functions of the associative network layer, and the top layer consists of a system of views which are special purpose windows into the knowledge base.

Keywords: knowledge base, associative networks
Computing Review Categories: I.2, I.6, H.2

Received August 1990, Accepted February 1991

1. The Need for Multiple Formalisms

The word "knowledge" will be used here to mean "a symbolic representation of aspects of some universe of discourse" [7]. A Knowledge Representation (KR) paradigm is a set of concepts and rules which can be used to formulate and rationalize one's knowledge [7]. The term KR formalism will also be used to refer to relatively well defined KR paradigms.

In the areas of databases, AI and programming languages, various KR formalisms have been put forward. For example entities and relationships, frames, rules, logic and objects. Many of these are intended to be general purpose formalisms. In practice it is found that each is useful for some application domains, but not so useful for others.

Sloman [8] states that there is no such thing as the ideal representation formalism. Many different kinds of objects need to be represented and for many different kinds of purposes. Therefore multiple formalisms are necessary [1]. There is also evidence that humans use a variety of paradigms involving images derived from different sensory modalities, and use different kinds of inference [8].

In industry there is often a need to use more than one formalism at the same time. There is an increasing need to fuse technologies from data bases, expert systems, hypertext, object oriented programming, CASE etc. [9,12,5].

An alternative to searching for the knowledge representation formalism of ultimate generality, could be to search for some medium (language and/or environment) that will make it easy to work with several formalisms at the same time, and that will make it easy to create or modify KR formalisms. The medium for implementing the various formalisms have so far been languages such as LISP or Prolog, or environments such as KEE [6]. The aim of this work has been to construct a medium that facilitates the implementation and usage of multiple formalisms. Such an environment should consist of:

1. A knowledge repository where different kinds of knowledge may be stored, irrespective of formalism.
2. A language that facilitates the implementation of various formalisms on the same knowledge base
3. A set of extensible tools and user interfaces suitable for creating and using multi-paradigm knowledge.

This environment should at least provide a common basis for implementing the main paradigms currently used:
(a) Records in files or relations
(b) Frames, semantic networks
(c) Production rules
(d) Hypermedia
(e) Objects in OOP and Object Oriented Databases

2. The Concept Network Environment

2.1 The Basic Structure

The aim of this work has been to construct an environment that meets the above requirements. Its design is based on the following central idea:
- Use an associative network, devoid of semantics, as a common basis to store knowledge from different paradigms.
- Provide a suitable language to easily construct semantic functions which interpret the associative network according to the different formalisms.

This environment consists of the following three layers:
2.2 The Associative Network Layer

A-Net consists of an associative network structure and a set of primitive functions that can be used to manipulate it. The motivation for using an associative network is based on the fact that some kind of labelled, directed graph has been found useful for representing knowledge in almost every discipline. In computer science too it is possible to think of the different semantics of different paradigms as coexisting on the same medium: the basic associative network. In the semantic layer, knowledge from different paradigms can be used as required by the problem, knowledge from one paradigm can freely use knowledge from another, certain pieces of knowledge can have different meanings in different contexts, and new KR formalisms can be created and existing ones extended. Thus structure and storage of knowledge are separated. The different layers will now be examined in more detail.

The bottom layer is A-Net: This is an associative network. The middle layer is the Semantic Layer: Here the semantics of different paradigms are stored in the form of semantic functions. Each semantic function interprets what is stored in A-Net in a manner that embodies a certain meaning. A-Net primitives are used by the different semantic functions to access the knowledge in A-Net.

The top layer is the Views layer: This is a system of special purpose windows, each with tools to view and manipulate the knowledge base below.

In this way knowledge from different paradigms coexists on the same medium: the basic associative network. In the semantic layer, knowledge from different paradigms can be used as required by the problem, knowledge from one paradigm can freely use knowledge from another, certain pieces of knowledge can have different meanings in different contexts, and new KR formalisms can be created and existing ones extended. Thus structure and storage of knowledge are separated. The different layers will now be examined in more detail.

The following are examples of the A-Net primitives. These functions can be used to create and manipulate the associative network.

\_Internalize(input): Finds the terminal node that represents input (e.g. the string "John") and returns the ID of that node. Similarly for entities other than character strings, such as file handles, remote addresses, images etc., external programs, etc. Function \_InternalizeN can create the terminal node if it does not exist.

\_Externalize(id): Given id (the ID of a terminal node), returns the physical output that the node represents.

\_NewNode(): This creates a new (internal) node and returns its ID.

\_Link(src, lbl, dest): Links node of ID src to the node of ID dest through an association labelled lbl. The arguments src and dest are integers and the argument lbl is a string.

\_Sources(dest): Returns a list containing the IDs of all source nodes linked through any links of any label to the given node of ID dest.

\_Sourcesl(dest, lbl): This is like \_Sources() but considers only links with label lbl. \_Sourcesl(dest, lbl) returns a single node ID, linked through a link labelled lbl.

\_Destinations(src): Returns a list containing the IDs of all destination nodes such that there is a link of any label from the given node of ID src to each of the destination nodes. \_Destinationsl(src, lbl) returns a list of only the destination nodes linked through links labelled lbl. \_Destinationsl(src, lbl) returns a single node linked through a link labelled lbl.

\_DestLabels(src): Returns all labels of links starting at the node of ID src.

\_SourceLabels(dest): Returns all labels of links ending at destination node of ID dest.

\_Search(condition): Searches through all terminal nodes and returns a list of ID's of nodes whose physical elements satisfy the condition. For example, search for all strings containing "oh" would return a list that includes the terminal node representing the string "John".

These and other A-Net primitives are used in the semantic layer to implement various semantic functions. Semantic functions can be written in any host language, such as Pascal or C, as long as the A-Net primitives can be used to manipulate the associative network.
The basic control structures and constructs of common procedural languages can be used for writing semantic functions:

- IF_THEN_ELSE_ENDIF
- WHILE_ENDWHILE
- Function calls for calling A-Net primitives and user-defined functions.
- Function definitions: The ability to define new functions.
- Assignment of values to variables.
- Data types: the most useful of which are integers, lists, character strings.
- The ability to store function identifiers as data and to execute at runtime (e.g. the C language's pointers to functions).

Some utility functions for handling arrays or lists are also useful.

- _ListMember(list, member): Returns TRUE or FALSE depending on whether member is in the list.
- _MaxList(list): Returns the number of elements in the list.
- _ListElement(list, elementnumber): Returns an element of the list.
- _AppendList(list, element): Returns the list with the element appended at the end.
- _ConcatList(list1, list2): Returns a new list consisting of list1 and list2 concatenated.

### 2.3 The Semantic Layer

In this layer each knowledge representation formalism is defined as a set of semantic functions. These functions interpret and manipulate the associative network below in different ways according to different formalisms.

Although each formalism is different from others, there is one common idea that will be used throughout. That is the idea of a concept and the idea of association of concepts into a concept network.

**The concept**: We will call a "concept", anything identifiable by the human mind. A concept is a single idea whether it is simple or complex.

Using terms from everyday life, a concept may represent a specific object or event in the "real" world or an imaginary object or event in the human mind. It may represent a well defined idea like a recipe for apple pie or an abstract idea such as freedom. It may represent a simple physical thing like a red dot, or a complex physical thing such as a building.

In using terms from different existing KR paradigms, examples of a concept would be a database record, a program, a frame, a rule, a hypertext document, a picture, a sound, a constraint, an object, a method, an entity, an entity set, a relationship, an event.

**The structure of a concept**: Each concept, such as the above, is a mental structure. By "structure" we mean that it consists of components placed in specific arrangements. This may be the arrangement of names in a list, or visual elements in a particular picture, or arrangement of events in time, etc. In the simplest case the structure may consist of a single concept, but in most cases the concept is a composite whole. The components are those parts of the concept that are of enough interest to warrant being identified as concepts in their own right. These are not components in a physical sense, and they are not necessarily arranged in space or time - they are semantic components. Therefore a concept is a semantic structure containing semantic components. The same components can be arranged differently to form a different concept. The arrangement is itself a semantic component of the concept.

**The concept network**: A concept is represented by a node in the associative network. Its associations to its semantic components are represented by links to the corresponding nodes. A set of semantic functions interpret and manipulate the concepts and their associations. This is the concept network.

Some examples will now be given of how existing formalisms can be represented in this framework: frames and semantic nets, production rules, relational databases, hypertext and OOP objects.

**Frames and semantic nets**: Each frame is a concept, and its slots are semantic components. Aspects of slots such as generic properties, default values, and slot conditions are semantic components of slots. All these identifiable concepts constituting a frame are represented by A-Net nodes. In writing semantic functions it should be remembered that these can be implemented in different ways.

Figure 1 shows an A-Net structure that can represents frames.

![Figure 1 An example of an A-Net frame structure.](image)

This example contains a generic frame "Person", a generic sub-frame "Employee" linked through an "ako" association, and an instance of a specific person "John" linked through an "isa" association to "Employee".

Some simple examples of semantic functions will be

---

*SACJ/SART, No 5, 1991*
presented below in order to give a general feel of the syntax and to illustrate how the A-Net primitives are used.

The following is an example (in pseudocode) of a semantic function that can be used to create generic frames:

```plaintext
CreateFrame(fname, slotlist)
{
  i=1
  frameno= NewNode()
  _Link(frameno,"framename",_InternalizeN(fname))
  WHILE i<=_MaxList(slotlist)
    x= NewNode() 
    _Link(frameno,_ListElement(slotlist,i),x)
    _Link(x,"default",_InternalizeN(_ListElement(slotlist,i+1)))
    _Link(x,"condition",_InternalizeN(_ListElement(slotlist,i+2)))
    i=i+3
  ENDWHILE
  RETURN framenode
}
```

Parameters: fname is the name of the frame, and slotlist is a list consisting of attribute name, default value and slot condition, repeated for each attribute. Demons could be added as a further aspect of slots.

Using this function, the generic frame "Person" could be created using the statement:

```plaintext
CreateFrame("Person","nameAnyone name!= blank age 30 0 <age< 100")
```

The statement frameno= NewNode() creates a new internal node which will represent the frame and stores its ID (say 123) in the variable framenode. The next statement creates a new terminal node which represents the string "Person" and links it to the frame node through a link labelled "framename". The following loop processes the attribute slots similarly. Finally it returns the ID of the node that represents the generic frame concept.

Throughout all semantic functions, concepts are referred to by their IDs. Here is a simple function that identifies the frame ID, given the frame's name:

```plaintext
Frame(fname)
{
  RETURN_SourceL(_Internalize(fname),"framename")
}
```

A function that declares the class/subclass relationship ("a kind of") between generic frames (given their IDs) can be the following:

```plaintext
Ako(subframe,superframe)
{
  _Link(subframe,"ako",superframe)
}
```

Using the above, the simple generic frame "Employee" (figure 1), can be created and classified as follows:

```plaintext
Ako(CreateFrame("Employee",""),Frame("Person")).
```

Similarly a function to create instances can be constructed, as can other functions.

Note that the similarity between Semantic Nets and A-Net is superficial. There are many types of semantic nets [2], each with different semantics. A-Net, on the other hand, has no KR semantics and can be used as a medium for their implementation.

**Rules:** When we think of a rule [7], we have a mental picture of it as consisting of a condition and a conclusion or action:

```plaintext
IF condition
THEN action
```

As in the case of frames, each rule is a concept. Its semantic components are its name, its conditions, and actions. Figure 2 illustrates a set of rules. These share some conditions or actions, thus forming rule chains.

Here is an example of a semantic function to create (simple) rules.

```plaintext
CreateRule(rname, statement, trueact)
{
  rule= NewNode()
  _Link(rule,"rulename",rname)
  _Link(rule,"condition",statement)
  _Link(rule,"action",trueact)
  RETURN rule
}
```

Backward chaining, the essence of many expert systems can be encoded in a semantic function such as the following:

```plaintext
BackChain(statement)
{
  rules=NULL_LIST
  IF IsTrue(statement)
  RETURN TRUE
  ELSE
    rules= SourcesL(statement,"action")
    IF NonEmpty(rules)
      rp=1
      WHILE rp <= _MaxList(rules)
        condition1 = DestinationL(_ListElement(rule,rp),"condition")
        IF IsTrue(condition1)
          RETURN TRUE
        ELSE
          IF BackChain(condition1)
            RETURN TRUE
          ELSE
            rp=rp+1
          END IF
        END IF
      ENDWHILE
      IF
      END IF
      ELSE
        IF BackChain(statement)
          RETURN TRUE
        ELSE
          END IF
        END ELSE
    END IF
}
```

Note that the similarity between Semantic Nets and A-Net is superficial. There are many types of semantic nets [2], each with different semantics. A-Net, on the other hand, has no KR semantics and can be used as a medium for their implementation.

**Rules:** When we think of a rule [7], we have a mental picture of it as consisting of a condition and a conclusion or action:

```plaintext
IF condition
THEN action
```

As in the case of frames, each rule is a concept. Its semantic components are its name, its conditions, and actions. Figure 2 illustrates a set of rules. These share some conditions or actions, thus forming rule chains.

Here is an example of a semantic function to create (simple) rules.

```plaintext
CreateRule(rname, statement, trueact)
{
  rule= NewNode()
  _Link(rule,"rulename",rname)
  _Link(rule,"condition",statement)
  _Link(rule,"action",trueact)
  RETURN rule
}
```

Backward chaining, the essence of many expert systems can be encoded in a semantic function such as the following:

```plaintext
BackChain(statement)
{
  rules=NULL_LIST
  IF IsTrue(statement)
  RETURN TRUE
  ELSE
    rules= SourcesL(statement,"action")
    IF NonEmpty(rules)
      rp=1
      WHILE rp <= _MaxList(rules)
        condition1 = DestinationL(_ListElement(rule,rp),"condition")
        IF IsTrue(condition1)
          RETURN TRUE
        ELSE
          IF BackChain(condition1)
            RETURN TRUE
          ELSE
            rp=rp+1
          END IF
        END IF
      ENDWHILE
      IF
      END IF
      ELSE
        IF BackChain(statement)
          RETURN TRUE
        ELSE
          END IF
        END ELSE
    END IF
}
```
IsTrue(statement) can be a semantic function that establishes the truth of a statement. If in the formalism being implemented here, establishing the truth of a statement meant that one would look it up in some fact base, then the fact base would be implemented as follows: Each fact (which is a concept) can be represented by a node and each such node is associated through a link labelled "truthtype" to a terminal node representing the string "Fact". To establish all the facts one would start at the node "Fact" and find all nodes associated to it. To establish if a given statement is a fact, one would start at the statement and follow its "truthtype" link to see if it is associated to node "Fact" (figure 2).

In a different formalism, IsTrue(statement) may mean that one should establish whether or not the statement is true by evaluating a given boolean expression based on the slot values of some frame. In such a formalism IsTrue() would follow a "truthvalue" link from the statement to an associated frame (as shown in figure 2) and use its slots to evaluate the boolean expression.

Similarly other formalisms can be implemented within the Rules paradigm (including the use of certainty factors). These may be implemented as alternatives or they can coexist without interfering with each other, since each semantic function uses the links and nodes that are meaningful to it and ignores the rest.

Function Establish(statement) can be a semantic function defined so as to ask the user the truth of a statement. TRUE and FALSE are the boolean constants of the host language.

Along similar lines ForwardChain(statement) or various other rule interpreters of A-Net can be constructed.

Examples of two paradigms have been presented above - frames and rules. The following paragraphs give a brief indication of how other formalisms are handled.

*Databases:* [4]. Tables consisting of records form the basic structure of many databases. The semantic functions to interpret A-Net in terms of a database formalism can be constructed similarly to those for frames.

Data models (distinct from the database itself) or constraints can also be implemented on A-Net. For example, an E-R diagram [3] as a whole is a concept. Its semantic components are the entity sets, relationships, cardinalities, etc. Semantic functions that interpret E-R schemas can be used during the construction of the different relational tables, or during the creation or change of the individual records (instances).

*Hypermedia:* Figure 3 illustrates an A-Net implementation of hypermedia. Functions such as EditText() and DrawFigure() residing at the terminal nodes, are functions, external to A-Net, for editing a piece of text or diagram. Semantic functions can be constructed to provide the basic navigational functions of hypermedia.

*Objects* as in OOP and Object Oriented Databases: Object classes, instances, encapsulation, and inheritance can be implemented using A-Net in a similar way as frames where implemented above. Methods are functions that reside on terminal nodes to which nodes representing class objects can be linked, as in hypermedia (figure 3).

A multi-paradigm knowledge base: Since A-Net was used throughout to implement the major formalisms, objects of the above formalisms can coexist on the same concept network. Using common storage for heterogeneous knowledge opens various possibilities:

Firstly, referencing by objects of one formalism to objects of another, is done easily and naturally by forming or breaking associative links, which do not interfere with the working of the specific semantic functions that ignore these links. For example, rules can be used to guide the navigation process through a hypermedia network, or the interaction with a database.

Secondly, possibilities are also open for viewing the same stored knowledge in different ways. For example, a set of rules can be used by semantic functions interpreting the knowledge as rules. At a certain point one may be interested in searching for all rules with conditions that involve age. In this case the same knowledge is being interpreted as records in a database. One may also want to classify the rules by linking them into a classification hierarchy without interfering with them in any other way. These, and other possibilities, are actually in the realm of Knowledge Representation. Our interest here is in providing the tool (A-Net) and the conceptual framework to facilitate this approach in KR and Knowledge Engineering.
2.4 The Views Layer
The next layer on top of the semantic layer is the views layer. This consists of a system of views. Each view is a special-purpose window used to view or modify the knowledge base. Each view uses semantic functions from the middle layer to extract or modify parts of the knowledge and to present this in a form which can best be understood by the user according to the specific task at hand.

Examples of views are a frame editor, a rule editor, a hypertext editor, an associative network browser, a record form layout editor and a query language.

3. Implementation
There are so far two implementations of A-Net: One (high speed/low volume) implementation of A-Net is memory resident, and was used to implement a design environment for semantic data models based on the Taxis language [10].

Another (low speed/high volume) implementation of A-Net is disk based on top of a relational database. This is an environment (named CoNET) and offers database, hypermedia, and expert system shell facilities, unified within the concept network. A commercial version of CoNET is now used in diverse applications such as law and software engineering [11].

4. Relation to Other Work
Although the above implementations aim to show the viability of this approach, more extensive application in different domains is needed in order to determine its value against other approaches. At this stage it is presented here as an alternative approach, which seems promising enough to pursue further. In comparing the use of A-Net with the use of other languages or systems, the following points can be of considered:

How does using A-Net differ from implementing the above formalisms in LISP or Prolog? The basic construct of LISP is lists and functions, while that of Prolog is logic and tree processing. In A-Net the basic idea is the concept network.

When using A-Net (within a host language or in a stand-alone interpreter) the emphasis is in dealing with multiple paradigms with uniform ease, so the basic structure aims to support this. For example, given a list, the LISP language can directly provide a certain element in the list, but not the reverse (special functions have to be written). In A-Net, the list as a whole, as well as each element of the list, are concepts represented by nodes, and primitives provide direct access in either direction. In Prolog it is easy to implement backward chaining, but forward chaining is more complex. Using A-Net, the latter is not more difficult.

"Hybrid" environments (shells) exist where knowledge can be encoded in two or more KR formalisms, such as rules and frames [6]. These environments integrate two or more specific formalisms so that they work together in the same environment. A-Net on the other hand attempts to unify the different paradigms by first identifying the associative network as the "highest common factor" of the formalisms and then using this as a medium to build each formalism on top of a common knowledge repository.

5. Future Work
Future work will consist of (1) formalizing A-Net further, (2) expanding CoNET with a larger library of formalisms and views, and (3) applying CoNET to problem areas requiring a multi-formalism approach. (4) Comparing to other approaches

6. Conclusions
Many problems require more than one knowledge representation paradigm. There is a need for multi-paradigm knowledge bases.

In essence the approach described here, separates the storage of knowledge from its structure: Instead of storing knowledge in a structure that is tailored for a specific formalism, all knowledge is stored in the associative network as a common medium. Structure itself, is moved to a separate level where it is implemented by semantic functions. This provides a flexible implementation medium, and a common conceptual framework for dealing with knowledge using different paradigms.
References

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:


Manuscripts accepted for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a typed form (i.e. suitable for scanning);
- as an ASCII file on diskette; or
- as a WordPerfect, TEX or LATEX or file; or
- in camera-ready format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, TEX or LATEX or file.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.
## Contents

**GUEST CONTRIBUTION**

Why all the Fuss About Neural Networks?
G Barth ........................................... 4

**RESEARCH ARTICLES**

The Placement of Subprograms by an Automatic Programming System
J P du Plessis and H J Messerschmidt .............................. 9

An Investigation into the Separation of the Application from its User Interface
J H Greyling and P R Warren .................................. 16

The Physical Correlates of Local Minima
L F A Wessels, E Barnard and E van Rooyen .................... 22

An Efficient Primal Simplex Implementation for the Continuous 2-Matching Problem
T H C Smith, T W S Meyer and L Leenen ......................... 28

Concept Network Framework for a Multi-paradigm Knowledge Base
J Kambanis ........................................ 32

PEST - A Microcomputer Pascal Based Expert System Shell
A G Sartori-Angus and R Neville .................................. 39

A Linda Solution to the Evolving Philosophers Problem
S E Hazelhurst ....................................... 44

**TECHNICAL NOTE**

Knowledge Representation using Formal Grammars
S H von Solms, E M Ehlers and D J Enslin ......................... 54

**COMMUNICATIONS AND REPORTS**

Book Review & Books Received ................................... 57

Cost-effective Visual Simulation Based on Graphics Workstations
A Caduri and D G Kourie ...................................... 58

A Method of Controlling Quality of Application Software
T D Crossman .......................................... 70

An Update on UNINET-ZA: The Southern African Academic and Research Network
V Shaw ............................................. 75