

# Integrating the Structured Analysis and Design Models: A Category-Theoretic Approach

T. H. Tse†

A number of models have been proposed under the name of structured analysis and design. It has been pointed out, however, that there is no common theoretical framework among them. Transformation of a specification from one model to another, although often recommended by authors, can only be done manually. A category-theoretic approach is proposed in this paper. As a result, development of structured specifications can be assisted through structured tasks and morphisms, and the integration of structured models can be achieved through functors and free categories.

Keywords and Phrases: **Keywords and phrases:** category theory, structured analysis, structured design

CR Categories: **D. 2. 1, D. 3. 1, G. 2, K. 6. 3**

## 1. INTRODUCTION

A number of models have been proposed under the name of structured analysis and design. Examples are data flow diagrams (DeMarco, 1978; Gane and Sarson, 1979; Weinberg, 1980), Jackson structure diagrams, Jackson structure text (Jackson, 1975), system specification diagrams, system implementation diagrams (Jackson, 1983), Warnier/Orr diagrams (Orr, 1977) and structure charts (Yourdon and Constantine, 1979). They are widely accepted by practising systems analysts and designers through the simplicity of use and the ease of communication with users. It has been pointed out by Tse (1986), however, that there is no common theoretical framework among them. Transformation of a specification from one model to another, although often recommended by authors, can only be done manually. An initial algebra approach has been proposed in that paper to integrate these models. Given a specification in one structured model, the initial algebra approach provides a formal means of mapping it to an equivalent specification in terms of another model. It does not, however, provide a means of developing the specification in the first place.

A category-theoretic alternative is proposed in this paper. The concept of an elementary task is defined.

---

Copyright © 1987, Australian Computer Society Inc.

General permission to republish, but not for profit, all or part of this material is granted, provided that the ACJ's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

† Centre of Computer Studies and Applications, University of Hong Kong, Pokfulam Road, Hong Kong. Manuscript received May, 1986; revised November, 1986.

Intuitively, it contains a process together with its input and output states. Structured tasks are defined as elementary tasks linked up by five operations: sequence, selection, parallelism, iteration and procedure call. The usual structured models such as DeMarco data flow diagrams, Yourdon structure charts or Jackson structure text can be seen as different ways of representing structured tasks. Morphisms, or functions preserving the structures, can be defined between tasks. The usual refinements in structured methodologies can be seen as the reverse of morphisms. These concepts help a system developer visualize the internal structure of a system, assemble or refine subsystems, and verify the consistency and completeness of a design.

Furthermore, we can integrate the models by providing mappings from one type of specification to another via the structured tasks. A DeMarco or Yourdon specification, for example, can be mapped to a structured task specification by means of a functor. Conversely, given a structured task specification, free categories of DeMarco or Yourdon specifications will result.

## 2. ADVANTAGES OF A CATEGORY-THEORETIC APPROACH

There are several advantages of using the category-theoretic approach to integrate the structured analysis and design models:

- a. Many attempts have already been made to computerize the system development environment. Examples are ISDOS (Teichroew and Hershey,

1977; Teichroew, 1980), SREM (Alford, 1985), EDDA (Trattnig and Kerner, 1980), UDS2 (Biggerstaff, 1979), SAMM (Stevens and Tripp, 1978) and USE (Wasserman, 1982). Most of these approaches, however, are developed independently of existing structured analysis and design models. As pointed out by Davis (1982) and Martin (1984), practitioners are rather hesitant to use such new tools because they involve an unfamiliar formal language. On the other hand, the category-theoretic approach simply adds a theoretical framework to existing structured models. If development environments are based on this concept, users can continue to employ the popular notations of structured specifications.

- b. Using the proposed approach, a system developer may conceive the target system in terms of structured tasks, which are the equivalents of standard structuring mechanisms such as sequence, selection and iteration used in the structured methodologies. The DeMarco, Yourdon or Jackson notations are simply seen as variations on the representation of the structured tasks. Morphisms or refinements can then be used by the system developer to manipulate the tasks. Properties of structured functions can be used to verify the correctness of such manipulation.
- c. The conversion of a specification from one model to another can be achieved through the concepts of functors and free categories. For example, a functor can map a DeMarco specification to a structured task. Furthermore, a structured task can be mapped to a number of Yourdon specifications. The system developer can then exercise discretion in choosing the appropriate specification according to system requirements.
- d. Different structured models are suitable for different situations depending on the environment (Shigo *et al.*, 1980), emphasis (Colter, 1982) and stage of development (Lauber, 1982). Several models are therefore required during the development process of one single system. If we provide system developers with a means of mapping one model to another, the efficiency of system development can be greatly improved.
- e. A number of development aids have been designed for individual structured models (Delisle *et al.*, 1982; DeMarco and Soceneantu, 1984; Tse, 1985). These aids, however, are useful only for individual models and are not applicable to others. If a way can be found to map one model to another, the development aid for one structured methodology may be applied to another.

### 3. A BRIEF INTRODUCTION TO CATEGORY THEORY

In this section, we will give a brief introduction to the fundamental concepts in category theory. Interested

readers may refer to Arbib and Manes (1975a; 1975b), Goguen *et al.* (1973; 1975; 1976), Goldblatt (1984) and Mac Lane (1971) for further details.

The concepts of categories and functors are stronger than the concepts of sets and functions in set theory. A *category* consists of a collection of *objects*, together with relationships amongst these objects called *morphisms*. Functions can be defined mapping objects in one category to those of another category. In particular, the most useful of these functions also preserve the morphisms amongst objects, and they are known as *functors*. More formally, a category  $X$  consists of

- a. a class of objects in  $X$ ;
- b. for each pair of objects  $A$  and  $B$  in  $X$ , a set of morphisms  $f: A \rightarrow B$

subject to the following conditions:

1. For each pair of morphisms  $f: A \rightarrow B$  and  $g: B \rightarrow C$ , there exists a morphism  $g \circ f: A \rightarrow C$ , which is called the composite of  $f$  and  $g$ .
2. Given any morphisms  $f: A \rightarrow B$ ,  $g: B \rightarrow C$  and  $h: C \rightarrow D$ ,  $h \circ (g \circ f) = (h \circ g) \circ f$ .
3. For each object  $A$  of  $X$ , there exists a morphism  $i_A: A \rightarrow A$ , called an identity morphism.
4. Given any morphism  $f: A \rightarrow B$ ,  $f \circ i_A = i_{f[A]} \circ f$ .

Mappings between categories preserving the morphisms are called functors. More formally, for any functor  $F: X \rightarrow Y$ ,

- a. each object  $A$  in  $X$  is related to one and only one object  $A$  in  $Y$ , denoted by  $A = F(A)$ ;
- b. each morphism  $f: A \rightarrow B$  in  $X$  is related to one and only one morphism  $F(f): F(A) \rightarrow F(B)$  in  $Y$

such that compositions and identities are preserved. That is to say,

1. if  $h = g \circ f$ , then  $F(h) = F(g) \circ F(f)$ ;
2.  $F(i_A) = i_{F[A]}$ .

### 4. STRUCTURED TASKS

Let us apply the categorical concepts to structured models. We define a *flow* as a tuple

$$\langle \langle s_{11}, s_{12}, \dots \rangle, p, \langle s_{21}, s_{22}, \dots \rangle \rangle$$

where  $p$  is a process,  $s_{11}, s_{12}, \dots$  are input states and  $s_{21}, s_{22}, \dots$  are output states. Any standard process  $p$  should have only one input state  $s_1$  and one output state  $s_2$ . The flow corresponding to a standard process will therefore be of the form  $\langle s_1, p, s_2 \rangle$ .

Besides the standard processes, we will also define auxiliary processes which may have more than one input or output states. There are three auxiliary processes:

- a. A *decision*, which has one input state  $s_1$  and two output states  $s_2$  and  $s_3$ . The flow corresponding to a decision is denoted by

- $\langle s_1, \oplus, \langle s_2, s_3 \rangle \rangle$ .
- b. A *collection*, which has two input states  $s_1$  and  $s_2$  and one output state  $s_3$ . The flow corresponding to a collection is denoted by  $\langle \langle s_1, s_2 \rangle, \oplus, s_3 \rangle$ .
- c. A *fork*, which also has one input state  $s_1$  and two output states  $s_2$  and  $s_3$ . The flow corresponding to a fork is denoted by  $\langle s_1, \otimes, \langle s_2, s_3 \rangle \rangle$ .
- d. A *join*, which also has two input states  $s_1$  and  $s_2$  and one output state  $s_3$ . The flow corresponding to a join is denoted by  $\langle \langle s_1, s_2 \rangle, \otimes, s_3 \rangle$ .

We define an *elementary task* as a set containing one and only one flow. A *task* in general is defined as either an empty set or a set containing one or more flows.

In structured models, we are more interested in tasks which are in the form of sequences, selections, iterations, parallelisms or procedure calls of other tasks. Such tasks are called *structured tasks*, and are defined recursively as follows:

- a. Any elementary task  $T = \{ \langle s_1, p, s_2 \rangle \}$  is structured. We will denote this structured task by  $T(s_1, s_2)$ .
- b. The *sequence* of any two structured tasks  $T_1(s_1, s_2)$  and  $T_2(s_2, s_3)$ , defined as  $(T_1 \bullet T_2)(s_1, s_3) = T_1(s_1, s_2) \cup T_2(s_2, s_3)$ , is a structured task.
- c. The *selection* of any two structured tasks  $T_1(s_1, s_2)$  and  $T_2(s_3, s_4)$ , defined as  $(T_1 \oplus T_2)(s_5, s_6) = \{ \langle s_5, \oplus, \langle s_1, s_3 \rangle \rangle \cup T_1(s_1, s_2) \cup T_2(s_3, s_4) \cup \{ \langle \langle s_2, s_4 \rangle, \oplus, s_6 \rangle \}$ , is a structured task.
- d. The *parallelism* of any two structured tasks  $T_1(s_1, s_2)$  and  $T_2(s_3, s_4)$ , defined as  $(T_1 \otimes T_2)(s_5, s_6) = \{ \langle s_5, \otimes, \langle s_1, s_3 \rangle \rangle \cup T_1(s_1, s_2) \cup T_2(s_3, s_4) \cup \{ \langle \langle s_2, s_4 \rangle, \otimes, s_6 \rangle \}$ , is a structured task.
- e. The *iteration* of any structured task  $T(s_1, s_2)$ , defined as  $*(T)(s_3, s_4) = \{ \langle \langle s_3, s_2 \rangle, \oplus, s_5 \rangle \cup T(s_1, s_2) \cup \{ \langle s_5, \oplus, \langle s_1, s_4 \rangle \rangle \}$ , is a structured task.
- f. The *procedure call* of any structured task  $T(s_1, s_2)$ , defined as  $\times(T)(s_3, s_4) = \{ \langle s_3, \otimes, \langle s_1, s_5 \rangle \rangle \cup T(s_1, s_2) \cup \{ \langle \langle s_2, s_5 \rangle, \otimes, s_4 \rangle \}$ , is a structured task.

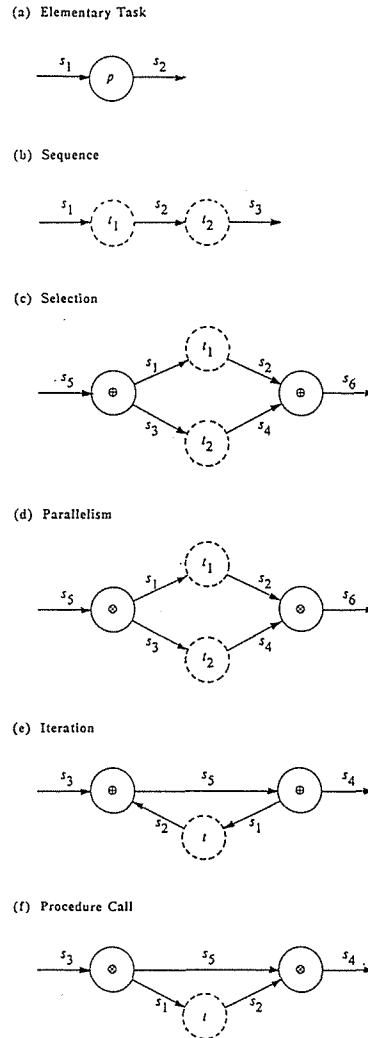


Figure 1. Diagrammatic Representation of Structured Tasks

To ease user understanding, we will represent structured tasks by task diagrams, as shown in Figure 1. Task diagrams are, in fact, a variation of DeMarco data flow diagrams. We simply forget about the input/output types and insist that every selection or parallelism must have a single input state and a single output state, as shown in the example of Figure 2.

### 5. STRUCTURED FUNCTIONS AND MORPHISMS

Given two tasks (or sets of flows)  $T_1$  and  $T_2$ , various functions  $f: T_1 \rightarrow T_2$  can be defined mapping individual flows of one task to those of the other. An important class of these functions is known as *structured functions*, which preserve the structuredness of the subtasks in  $T_1$  and  $T_2$ . Formally, a structured function  $f$  must satisfy the following condition for any subtask  $T$  in  $T_1$ :

$T$  is structured if and only if  $f[T]$  is structured.

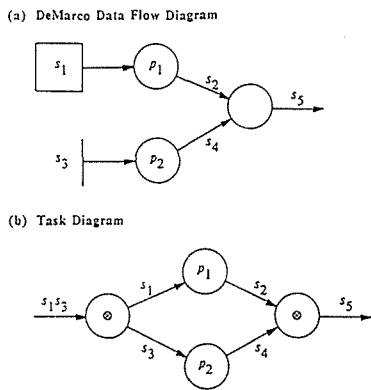


Figure 2. Example showing Relationship between DeMarco Data Flow Diagrams and Task Diagrams

We are interested in actually constructing a structured function. This can be done through abstractions and morphisms. A function  $f: T_1 \rightarrow T_2$  is an *abstraction* if and only if there exist a structured subtask  $T$  in  $T_1$  and an elementary subtask  $\{ \langle s_1, p, s_2 \rangle \}$  in  $T_2$  such that

- a. any flow in  $T$  is mapped to the single flow  $\langle s_1, p, s_2 \rangle$  in  $T_2$ ;
- b. any other flow in  $T_1$  is mapped to the same flow in  $T_2$ .

*Morphisms* are then defined between tasks using the following recursive definition:

- a. An abstraction is a morphism.
- b. Compositions of morphisms are morphisms.

It can be shown that any morphism must be a structured function. In other words, a morphism preserves the structures of the tasks.

We will relate the concept of morphisms to the usual concept of refinements in structured analysis and design. A task  $T_1$  is said to be a *refinement* of another task  $T_2$  if and only if there exists a morphism  $f$  mapping  $T_1$  on to  $T_2$ . In this way, morphisms and refinements help the users to visualize the development of a specification. The sample specification in Tse (1986), for instance, can be developed in terms of morphisms and refinements. Thus, using the obvious morphisms,

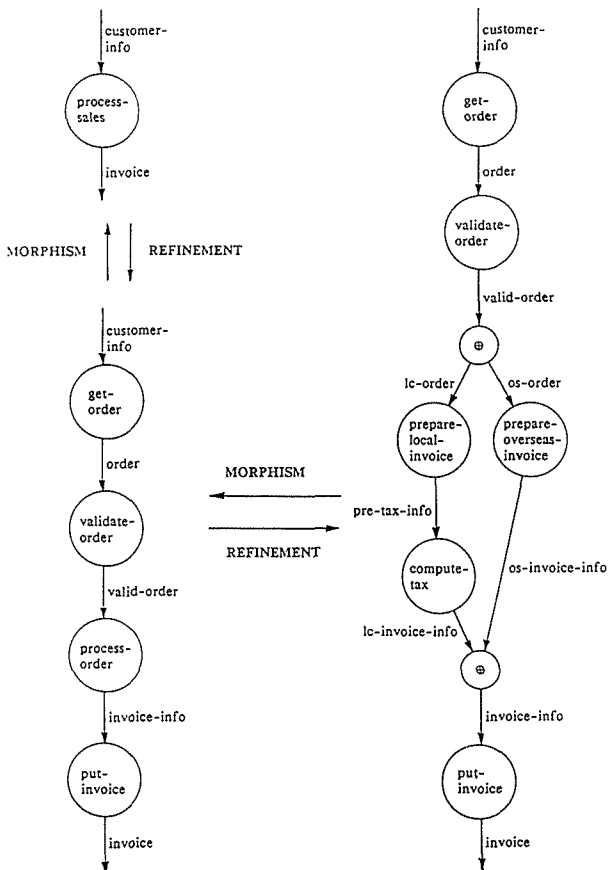


Figure 3. Diagrammatic Representation of Morphisms and Refinements

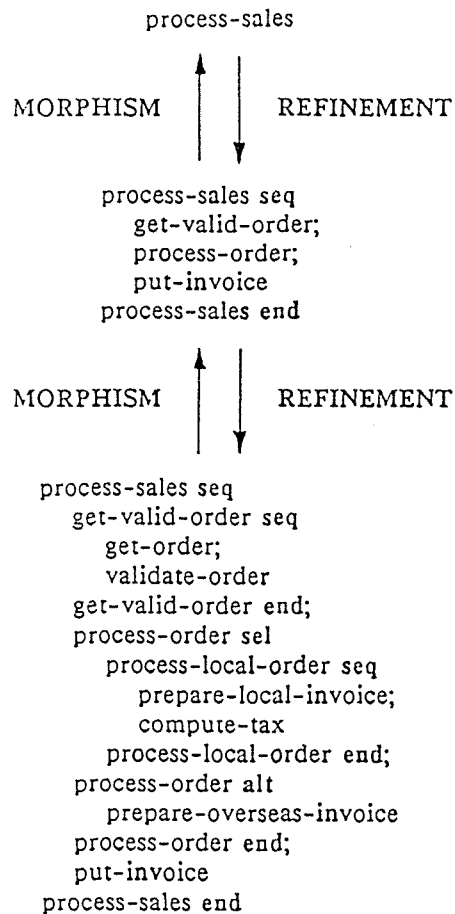


Figure 4. Example of Morphisms and Refinements in Jackson Structure Text

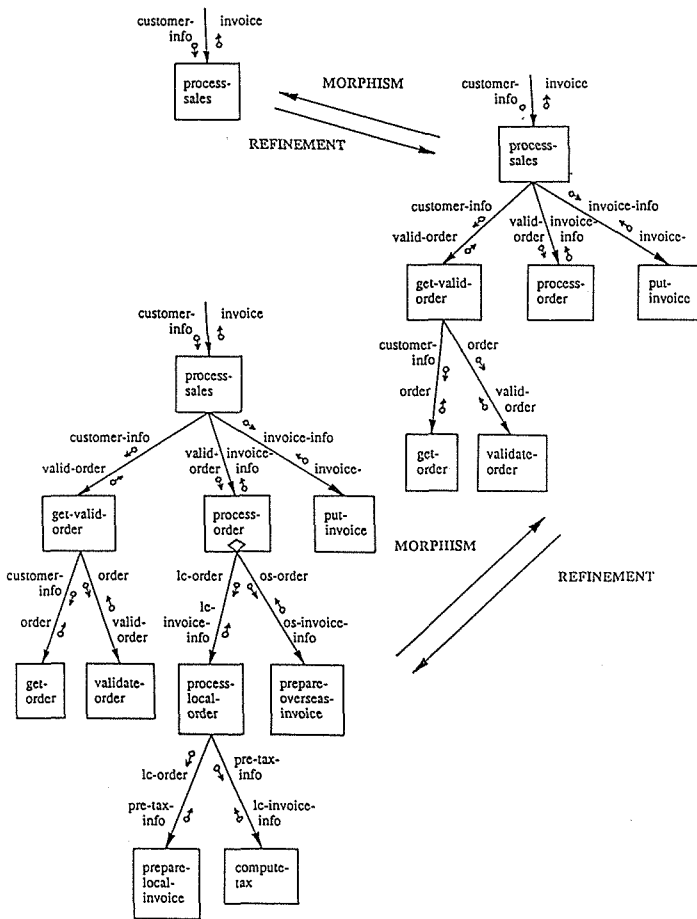


Figure 5. Morphisms and Refinements in Yourdon Structure Charts

$T_1 = \{ \langle \text{customer-info}, \text{process-sales}, \text{invoice} \rangle \}$   
 can be refined to

$T_2 = \{ \langle \text{customer-info}, \text{get-valid-order}, \text{valid-order} \rangle, \langle \text{valid-order}, \text{process-order}, \text{invoice-info} \rangle, \langle \text{invoice-info}, \text{put-invoice}, \text{invoice} \rangle \}$ ,

which can be further refined to

$T_3 = \{ \langle \text{customer-info}, \text{get-order}, \text{order} \rangle, \langle \text{order}, \text{validate-order}, \text{valid-order} \rangle, \langle \text{valid-order}, \oplus, \langle \text{lc-order}, \text{os-order} \rangle \rangle, \langle \text{lc-order}, \text{prepare-local-invoice}, \text{pre-tax-info} \rangle, \langle \text{pre-tax-info}, \text{compute-tax}, \text{lc-invoice-info} \rangle, \langle \text{os-order}, \text{prepare-overseas-invoice}, \text{os-invoice-info} \rangle, \langle \langle \text{lc-invoice-info}, \text{ls-invoice-info} \rangle, \oplus, \text{invoice-info} \rangle, \langle \text{invoice-info}, \text{put-invoice}, \text{invoice} \rangle \}$

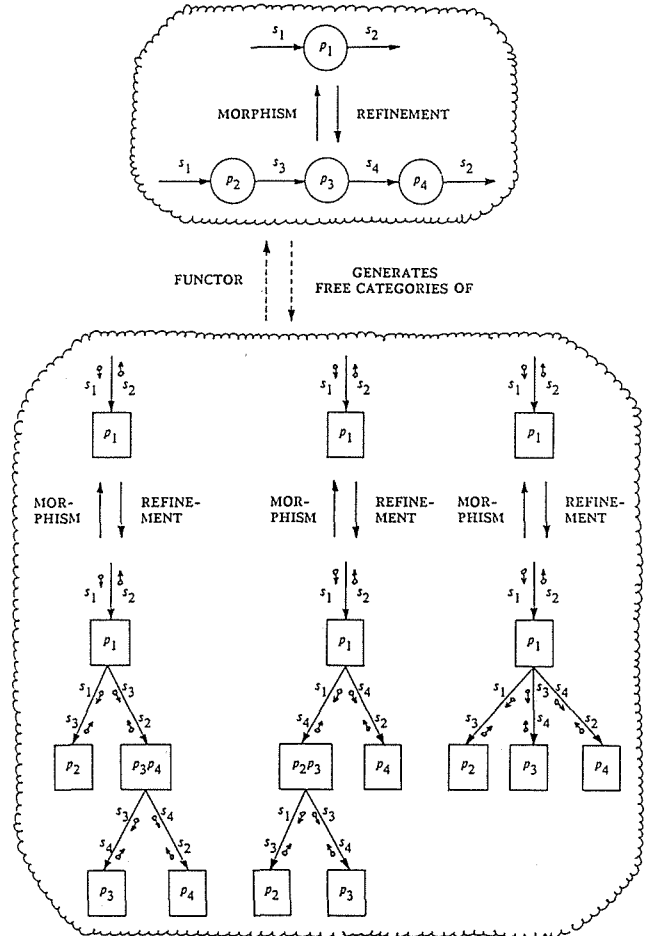


Figure 6. Illustration of Functors and Free Categories

and so on. In this way, stepwise refinement can simply be conceived as a set-manipulation process, which can easily be aided by a computerized system. For the sake of user friendliness, however, we can also represent the refinement steps diagrammatically, as shown in Figure 3.

## 6. FUNCTORS AND FREENESS

If we modify our structured tasks and morphisms slightly, we will obtain the common structured models. For example, if each of the input and output states is given a type such as source, sink, file or data, then the structured tasks can be represented diagrammatically by DeMarco data flow diagrams. If we forget about the names of input and output states, we can represent the structured tasks by Jackson structure text, as shown in Figure 4. If all the abstractions are documented, one level at a time, we can represent the structured tasks by Yourdon structure chart, as shown in Figure 5.

Since the structured models are slightly different ways of representing structured tasks, we would like to know whether there is any mapping which helps us convert one representation to another. It can be shown that the structured tasks and the morphisms form a category. Functors, or functions which preserve the morphisms, can be defined between categories. A Yourdon specification, for example, can be mapped to a task diagram specification by means of a functor. In other words, given a Yourdon specification, one and only one structured task specification will result. Conversely, a structured task specification can be mapped to free categories of Yourdon specifications. In this case, the system developer will be presented with possible design choices and may select the appropriate implementation based on personal experience, environmental considerations and hardware characteristics. Functors and free categories are illustrated in Figure 6.

The arguments for DeMarco and Jackson specifications are similar. As a result, the structured models can be linked up through the paths shown in Figure 7. Thus, we can integrate the models by providing categorical bridges from one type of specification to another. A DeMarco specification, for example, can be mapped to a number of Yourdon specifications via a structured task. The system designer can then exercise discretion in choosing the appropriate option according to system requirements.

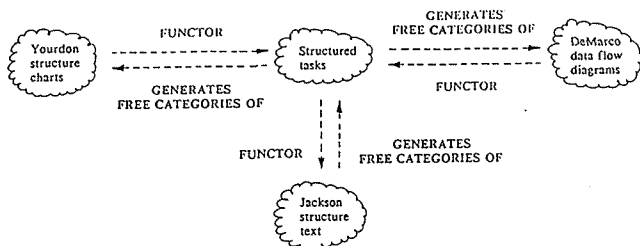


Figure 7. Categorical Relationships among Structured Models

## 7. CONCLUSION

A category-theoretic approach is proposed in this paper to link up the structured analysis and design models. The concepts of structured tasks and morphisms are defined. They help a system developer visualize the internal structure of a system, assemble or refine subsystems, and verify the consistency and completeness of a design. These can be done through simple set-manipulation with the aid of task diagrams.

The structured tasks and the morphisms form a category. Similar categories can be defined over other structured models such as DeMarco data flow diagrams, Yourdon structure chart and Jackson structured text. We can integrate the models by providing categorical bridges from one type of specification to another via the structured tasks. A DeMarco specification, for example, can be mapped to a number of Yourdon specifications. The system developer will be

presented with possible design choices and may select the appropriate implementation based on personal experience, environmental considerations and hardware characteristics.

## ACKNOWLEDGEMENTS

Part of this research was done at the London School of Economics, University of London under a Commonwealth Academic Staff Scholarship. The author is grateful to Mr Ronald Stamper, Dr Haya Freedman and Professor Frank Land of LSE, Dr Kit Grindley of Price Waterhouse and Dr Sami Zahran of ICL (UK) for their invaluable comments and suggestions. The research was also supported in part by a University of Hong Kong Research Grant.

## References

- ALFORD, M.W. (1985): SREM at the age of eight: the distributed computing design system, *IEEE Computer*, Vol. 18, No. 4, pp. 36-46.
- ARBIB, M.A. and MANES, E.G. (1975a): *Arrows, structures and functors: the categorical imperative*, Academic Press, New York.
- ARBIB, M.A. and MANES, E.G. (1975b): Basic concepts of category theory applicable to computation and control, *Lecture Notes in Computer Science*, Vol. 25, Springer-Verlag, Berlin, pp. 1-34.
- BIGGERSTAFF, T.J. (1979): The unified design specification system (UDS2), *Proceedings of Conference on Specifications of Reliable Software*, IEEE Computer Society, New York, pp. 104-118.
- COLTER, M.A. (1982): 'Evolution of the structured methodologies', in *Advanced System Development/Feasibility Techniques*, J.D. Couger, M.A. Colter and R.W. Knapp (eds.), Wiley, New York.
- DAVIS, A.M. (1982): The design of a family of application-oriented requirements languages, *IEEE Computer*, Vol. 15, No. 5, pp. 21-28.
- DELISLE, N.M., MENICOSY, D.E., and KERTH, N.L. (1982): 'Tools for supporting structured analysis', in *Automated Tools for Information System Design*, North-Holland, Amsterdam, H.-J. Schneider and A.I. Wasserman (eds.), pp. 11-20.
- DEMARCO, T. (1978): *Structured Analysis and Systems Specification*, Prentice-Hall, Englewood Cliffs, New Jersey.
- DEMARCO, T. and SOCENEANTU, A. (1984): SYNCRO: A dataflow command shell for the Lilith/Modula computer, *Proceedings of 7th International Conference on Software Engineering*, IEEE, New York, pp. 207-213.
- GANE, C. and SARSON, T. (1979): *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey.
- GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G., and WRIGHT, J.B. (1973): *A junction between computer science and category theory, I: Basic definitions and examples, Part 1*, IBM T.J. Watson Research Centre, Yorktown Heights, New York, Research Report RC-4526.
- GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G., and WRIGHT, J.B. (1975): *An introduction to categories, algebraic theories and algebras*, IBM T.J. Watson Research Center, Yorktown Heights, New York, Research Report RC-5369.
- GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G., and WRIGHT, J.B. (1976): *A junction between computer science and category theory, I: Basic definitions and examples, Part 2*, IBM T.J. Watson Research Centre, Yorktown Heights, New York, Research Report RC-5908.

- GOLDBLATT, R. (1984): *Topoi: the Categorical Analysis of Logic*, North-Holland, Amsterdam.
- JACKSON, M.A. (1975): *Principles of Program Design*, Academic Press, London.
- JACKSON, M.A. (1983): *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey.
- LAUBER, R.J. (1982): Development support systems, *IEEE Computer*, Vol. 15, No. 5, pp. 36-46.
- MAC LANE, S. (1971): *Categories for the Working Mathematician*, Springer-Verlag, New York.
- MARTIN, J. (1984): *An Information Systems Manifesto*, Prentice-Hall, Englewood Cliffs, New Jersey.
- ORR, K.T. (1977): *Structured Systems Development*, Yourdon, New York.
- SHIGO, O., IWAMOTO, K., and FUJIBAYASHI, S. (1980): A software design system based on a unified design methodology, *Journal of Information Processing*, Vol. 3, No. 3, pp. 186-196.
- STEVENS, S.A. and TRIPP, L.L. (1978): Requirements expression and verification aid, *Proceedings of 3rd International Conference on Software Engineering*, IEEE Computer Society, Silver Spring, Maryland, pp. 101-108.
- TEICHROEW, D. and HERSHEY, E.A. (1977): PSL/PSA: a computer-aided technique for structured documentation and analysis of information processing systems, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 41-48.
- TEICHROEW, D. (1980): 'Application of the entity-relationship approach to information processing systems modelling', in *Entity-Relationship Approach to Systems Analysis and Design*, P.P. Chen (ed.), North-Holland, Amsterdam, pp. 15-39.
- TRATTNIG, W. and KERNER, H. (1980): EDDA: a very-high-level programming and specification language in the style of SADT, *Proceedings of 4th International Software and Applications Conference*, IEEE Computer Society, Chicago, pp. 436-443.
- TSE, T.H. (1985): An automation of Jackson's structured programming, *Australian Computer Journal*, Vol. 17, No. 4, pp. 154-162.
- TSE, T.H. (1986): Integrating the structured analysis and design models: an initial algebra approach, *Australian Computer Journal*, Vol. 18, No. 3, pp. 121-127.
- WASSERMAN, A.I. (1982): 'The user software engineering methodology: an overview', in *Information Systems Design Methodologies: a Comparative Review*, North-Holland, Amsterdam, T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart (eds.), pp. 591-628.
- WEINBERG, V. (1980): *Structured Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.
- YOURDON, E. and CONSTANTINE, L.L. (1979): *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press, New York.

### **Biographical Note**

*T. H. Tse received his B. Sc. degree from the University of Hong Kong in 1970 and his M. Sc. degree from the University of London in 1979. He is currently a lecturer in computer science at the University of Hong Kong. His research interests include formal methods in information systems and software engineering.*

*Mr Tse is a member of the British Computer Society, the British Institute of Management and the Institute of Data Processing Management. He is a council member of the Vocational Training Council in Hong Kong. He was awarded an M.B.E. by the Queen in 1982.*

## **CALL FOR PAPERS FOR A SPECIAL ISSUE ON DIGITAL IMAGE PROCESSING**

The *Australian Computer Journal* will publish a special issue on 'Digital Image Processing' in August, 1987, under the guest editorship of John O'Callaghan of the CSIRO Division of Information Technology in Canberra. Research papers, tutorial articles and industry case studies on any aspects of the subject including computer vision, hand-eye systems, image restoration and reconstruction, image enhancement, image compression or image analysis will be welcome. Both full papers and short communications will be considered and prospective authors should write to, or telephone, as soon as possible:

**Dr J.F. O'Callaghan,  
CSIRO Division of Information Technology,  
PO Box 1800,  
Canberra City, ACT 2601.  
(Telephone (062) 433-204)**

to inform him of their intention to submit material for the issue and to provide a brief summary of their proposed contribution. In order to allow adequate time for refereeing and editorial review, complete manuscripts will be required by *20 March, 1987*. These should be prepared in accordance with the guidelines published in the February, 1986 issue of this Journal.