HKU CSIS Tech Report TR-A5-86

Australian Computer Journal **19** (1): 25−31 (1987)

Integrating the Structured Analysis and Design Models: a Category-Theoretic Approach [*] [†]

T.H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
Email: thtse@cs.hku.hk

ABSTRACT

Quite 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: category theory, structured analysis, structured design

CR categories: D.2.1, D.3.1, G.2, K.6.3

1. INTRODUCTION

Quite a number of models have been proposed under the name of structured analysis and design. Examples are data flow diagrams (DeMarco, 1979; 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

* © 1987 *Australian Computer Journal*. This material is presented to ensure timely dissemination of scholarly and technical work. Personal use of this material is permitted. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each authors copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from *Australian Computer Journal*.

† Part of this research was done at the London School of Economics, University of London under a Commonwealth Academic Staff Scholarship.

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. 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 (Stephens 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 shall 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:

(i) 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.

(ii) Given any morphisms $f: A \rightarrow B$, $g: B \rightarrow C$ and $h: C \rightarrow D$,

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

(iii) For each object A of X, there exists a morphism $i_A: A \rightarrow A$, which is called an identity morphism.

(iv) 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 with one and only one object A' in Y, denoted by $A' = F(A)$;

(b) each morphism $f: A \rightarrow B$ in X is related with 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,
   - if $h = g \circ f$, then $F(h) = F(g) \circ F(f)$;
   - $F(i_A) = i_{F[A]}$.

4. STRUCTURED TASKS

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

$$<<s_{11}, s_{12}, \ldots>, p, <s_{21}, s_{22}, \ldots>>,$$

where p is a process, s_{11}, s_{12}, ... are input states and s_{21}, s_{22}, ... 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 $<s_1, p, s_2>$.

Besides the standard processes, we shall also define auxiliary processes which may have more than one input or output states. There are four 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 $<s_1, \oplus, <s_2, s_3>>$.
(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 $<<s_1, s_2>, \oplus, s_3>$.
(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 $<s_1, \otimes, <s_2, s_3>>$.
(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 $<<s_1, s_2>, \otimes, s_3>$.

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 $T = \{ <s_1, p, s_2> \}$ is structured. We shall 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

$$(T_1 \oplus T_2)(s_5, s_6) = \{ <s_5, \oplus, <s_1, s_3>> \} \cup T_1(s_1, s_2) \cup T_2(s_3, s_4) \\ \cup \{ <<s_2, s_4>, \oplus, s_6> \},$$

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

$$(T_1 \otimes T_2)(s_5, s_6) = \{ <s_5, \otimes, <s_1, s_3>> \} \cup T_1(s_1, s_2) \cup T_2(s_3, s_4) \\ \cup \{ <<s_2, s_4>, \otimes, s_6> \},$$

is a structured task.

(e) The *iteration* of any structured task $T(s_1, s_2)$, defined as

$$*(T)(s_3, s_4) = \{ <<s_3, s_2>, \oplus, s_5> \} \cup T(s_1, s_2) \cup \{ <s_5, \oplus, <s_1, s_4>> \},$$

is a structured task.

(f) The *procedure call* of any structured task $T(s_1, s_2)$, defined as

$$\times(T)(s_3, s_4) = \{ <s_3, \otimes, <s_1, s_5>> \} \cup T(s_1, s_2) \cup \{ <<s_2, s_5>, \otimes, s_4> \},$$

is a structured task.

To ease user understanding, we shall 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 $f\colon T_1 \to T_2$ 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.

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

(a) any flow in T is mapped to the single flow $<s_1, p, s_2>$ 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) Composition 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 shall 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,

T_1 = { <*customer-info*, process-sales, *invoice*> }

can be refined to

T_2 = { <*customer-info*, get-valid-order, *valid-order*>,
<*valid-order*, process-order, *invoice-info*>,
<*invoice-info*, put-invoice, *invoice*> },

which can be further refined to

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

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 shall 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. An illustration of functors and free categories is shown 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.

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

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. (1975): *Arrows, Structures, and Functors: the Categorical Imperative*, Academic Press, New York, NY.

Arbib, M.A. and Manes, E.G. (1975): "Basic concepts of category theory applicable to computation and control", in *Category Theory applied to Computation and Control: Proceedings of the 1st International Symposium*, E.G. Manes (ed.), Lecture Notes in Computer Science, Springer, Berlin, Germany, Vol. 25, pp. 1−34.

Biggerstaff, T.J. (1979): "The unified design specification system (UDS^2)", in *Proceedings of the IEEE Conference on Specifications of Reliable Software*, M.V. Zelkowitz (ed.), IEEE Computer Society Press, New York, NY, 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, NY, pp. 73−96.

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 Systems Design*, H.-J. Schneider and A.I. Wasserman (eds.), Elsevier, Amsterdam, The Netherlands, pp. 11−20.

DeMarco, T. (1979): *Structured Analysis and System Specification*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

DeMarco, T. and Soceneantu, A. (1984): "SYNCRO: a data flow command shell for the Lilith/Modula computer", in *Proceedings of the 7th International Conference on Software Engineering* (*ICSE 1984*), IEEE Computer Society Press, New York, NY, pp. 207−213.

Gane, C.P. and Sarson, T. (1979): *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, Englewood Cliffs, NJ.

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", Research Report RC-4526, IBM T.J. Watson Research Center, Yorktown Heights, NY.

Goguen, J.A., Thatcher, J.W., Wagner, E.G., and Wright, J.B. (1975): "An introduction to categories, algebraic theories, and algebras", Research Report RC-5369, IBM T.J. Watson Research Center, Yorktown Heights, NY.

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", Research Report RC-5908, IBM T.J. Watson Research Center, Yorktown Heights, NY.

Goldblatt, R. (1984): *Topoi: the Categorial Analysis of Logic*, Elsevier, Amsterdam, The Netherlands.

Jackson, M.A. (1975): *Principles of Program Design*, Academic Press, London, UK.

Jackson, M.A. (1983): *System Development*, Prentice Hall International Series in Computer Science, Prentice Hall, London, UK.

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, New York, NY.

Martin, J. (1984): *An Information Systems Manifesto*, Prentice Hall, Englewood Cliffs, NJ.

Orr, K.T. (1977): *Structured Systems Development*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

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.

Stephens, S.A. and Tripp, L.L. (1978): "Requirements expression and verification aid", in *Proceedings of the 3rd International Conference on Software Engineering* (*ICSE 1978*), IEEE Computer Society Press, New York, NY, pp. 101−108.

Teichroew, D. and Hershey, E.A. III (1977): "Computer-aided software development", in *Software Reliability*, State of the Art Report, Infotech, Maidenhead, UK, Vol. 2, pp. 299−368.

Teichroew, D., Macasovic, P., Hershey E.A. III, and Yamamoto, Y. (1980): "Application of the entity-relationship approach to information processing systems modelling", in *Entity-Relationship Approach to Systems Analysis and Design: Proceedings of the 1st Conference on Entity-Relationship Approach*, P.P. Chen (ed.), Elsevier, Amsterdam, The Netherlands, pp. 15−39.

Trattnig, W. and Kerner, H. (1980): "EDDA: a very-high-level programming and specification language in the style of SADT", in *Proceedings of the 4th Annual International Computer Software and Applications Conference* (*COMPSAC 1980*), IEEE Computer Society Press, New York, NY, 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: Proceedings of the IFIP WG 8.1 Working Conference* (*CRIS 1982*), T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart (eds.), Elsevier, Amsterdam, The Netherlands, pp. 591−628.

Weinberg, V. (1980): *Structured Analysis*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

Yourdon, E. and Constantine, L.L. (1979): *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

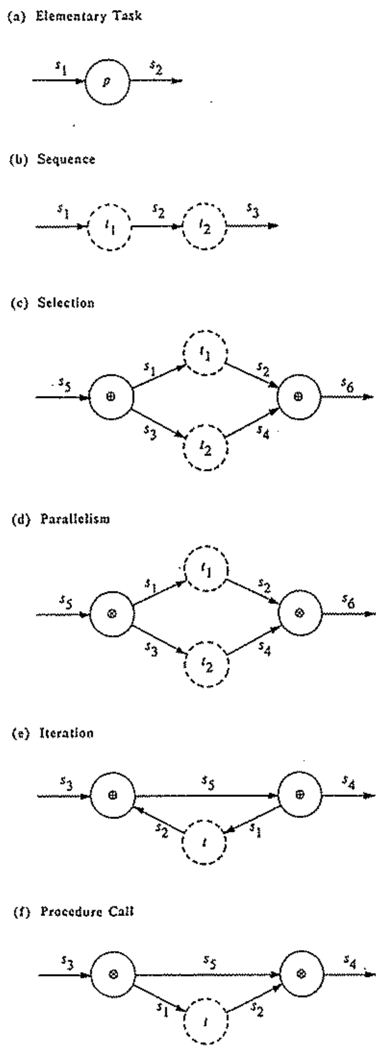

(a) Elementary Task
s_1 p s_2
(b) Sequence
s_1 t_1 s_2 t_2 s_3
(c) Selection
s_5 s_1 t_1 s_2 s_6 s_3 t_2 s_4
(d) Parallelism
s_5 s_1 t_1 s_2 s_6 s_3 t_2 s_4
(e) Iteration
s_3 s_5 s_4 s_2 t s_1
(f) Procedure Call
s_3 s_5 s_4 s_1 t s_2

Figure 1. Diagrammatic Representation of Structured Tasks

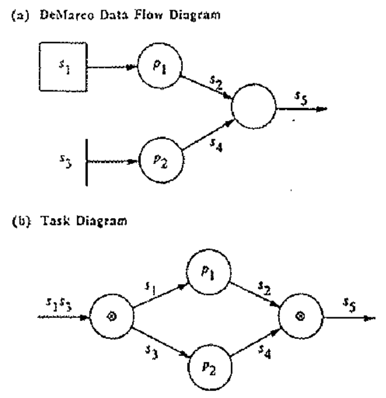


Figure 2. Example showing Relationship between DeMarco Data Flow Diagrams and Task Diagrams

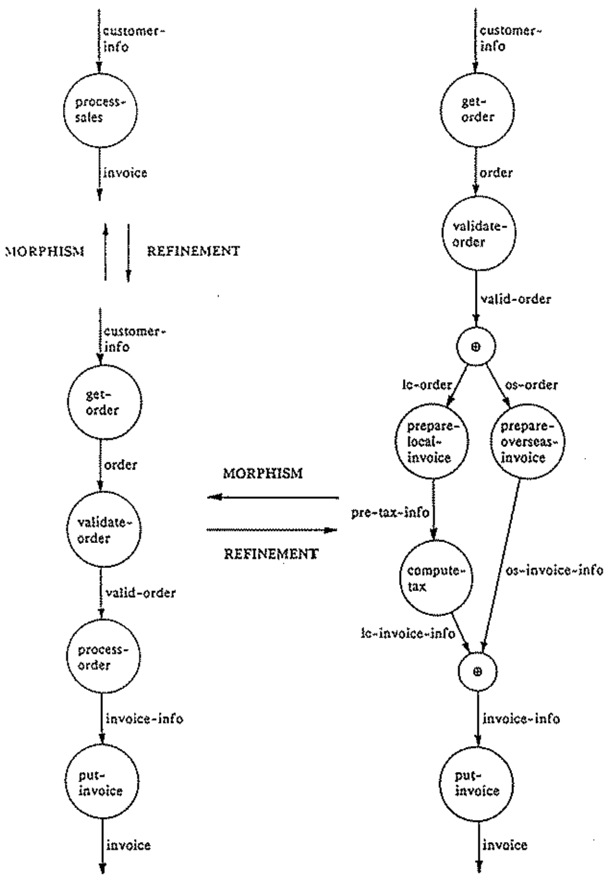
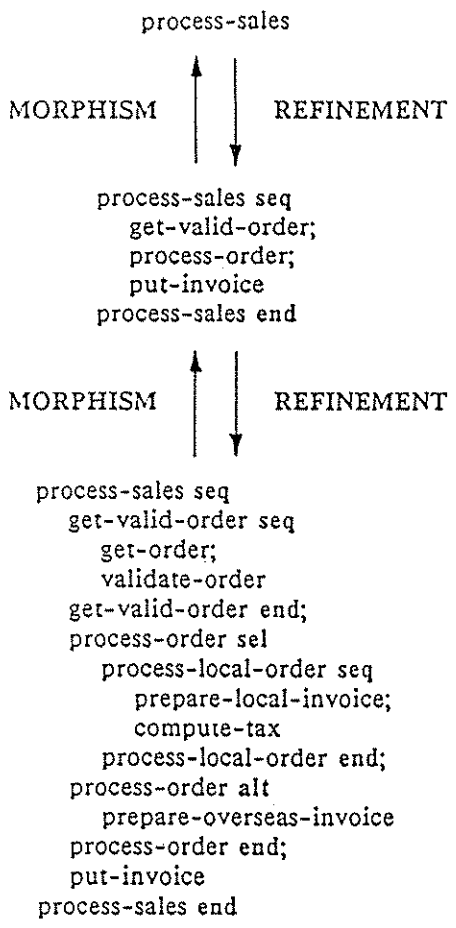


Figure 3. Diagrammatic Representation of Morphisms and Refinements



```
process-sales
```

```
process-sales seq
   get-valid-order;
   process-order;
   put-invoice
process-sales end
```

```
process-sales seq
   get-valid-order seq
      get-order;
      validate-order
   get-valid-order end;
   process-order sel
      process-local-order seq
         prepare-local-invoice;
         compute-tax
      process-local-order end;
   process-order alt
      prepare-overseas-invoice
   process-order end;
   put-invoice
process-sales end
```

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

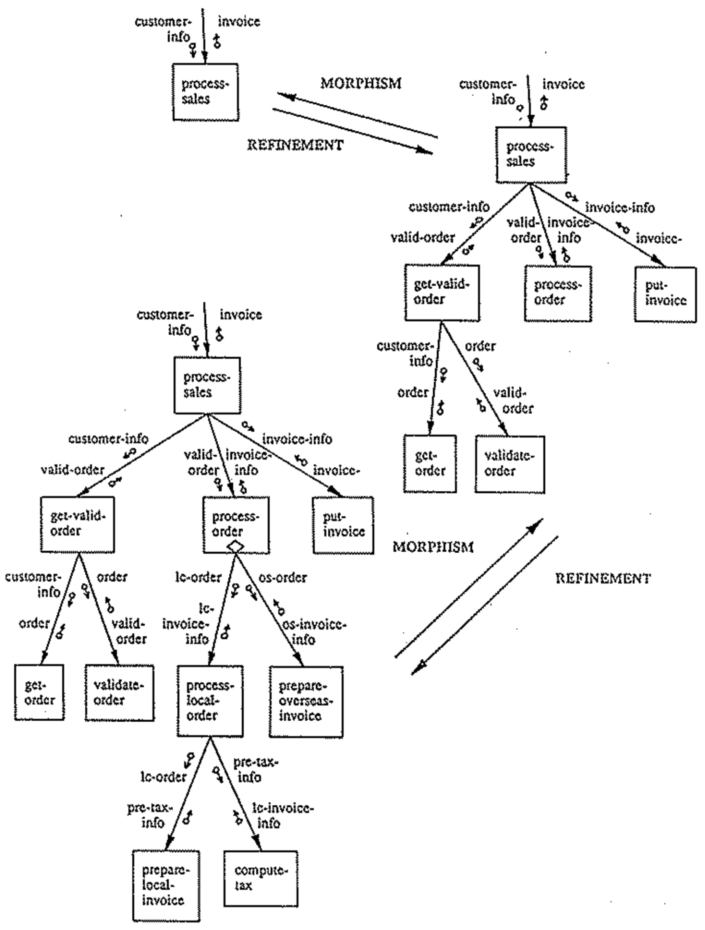


Figure 5. Morphisms and Refinements in Yourdon Structure Charts

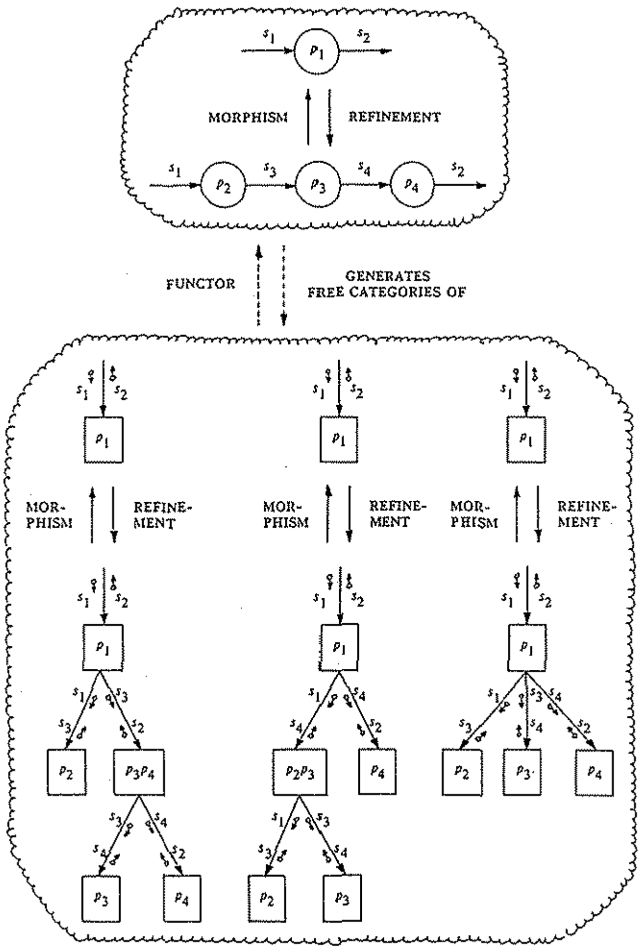


Figure 6. Illustration of Functors and Free Categories

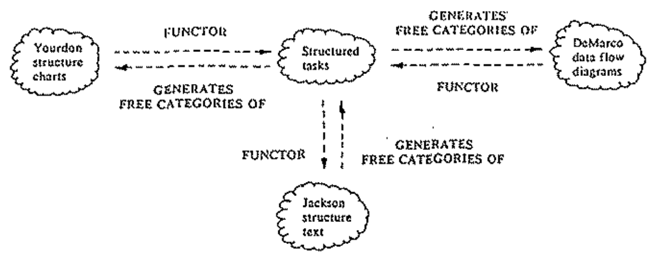

Yourdon structure charts
FUNCTOR
GENERATES FREE CATEGORIES OF
Structured tasks
GENERATES FREE CATEGORIES OF
FUNCTOR
DeMarco data flow diagrams
FUNCTOR
GENERATES FREE CATEGORIES OF
Jackson structure text

Figure 7. Categorical Relationships among Structured Models