A Unifying Framework for Structured Analysis and Design Models

T. H. Tse

Cambridge Tracts in Theoretical Computer Science 11

TP31
T882

A Unifying Framework for Structured Analysis and Design Models: An Approach using Initial Algebra Semantics and Category Theory

T.H. Tse, M.B.E.
University of Hong Kong

The right of the University of Cambridge to print and sell all manner of books was granted by Henry VIII in 1534. The University has printed and published continuously since 1584.

CAMBRIDGE UNIVERSITY PRESS
Cambridge
New York Port Chester Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
40 West 20th Street, New York, NY 10011, USA
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1991

Printed in Great Britain at the University Press, Cambridge

Library of Congress cataloguing in publication data available

British Library cataloguing in publication data available

ISBN 0 521 39196 2

To Teresa

Preface

Structured analysis and design methodologies have been recognized as a popular and powerful tool in information systems development. A complex system can be specified in a top-down and graphical fashion, enabling practitioners to visualize the target systems and communicate with users much more easily than by means of conventional methods. As a matter of fact, the structured methodologies have been designed by quite a number of distinct authors, each employing a number of models which vary in their graphical outlook. Different models are found to be suitable for different stages of a typical systems life cycle. A specification must be converted from one form to another during the development process. Unfortunately, however, the models are only derived from the experience of the authors. Little attempt has been made in proposing a formal framework behind them or establishing a theoretical link between one model and another.

A unifying framework is proposed in this book. We define an initial algebra of structured systems, which can be mapped by unique homomorphisms to a DeMarco algebra of data flow diagrams, a Yourdon algebra of structure charts and a Jackson algebra of structure texts. We also find that the proposed initial algebra as well as the structured models fit nicely into a functorial framework. DeMarco data flow diagrams can be mapped by a free functor to terms in the initial algebra, which can then be mapped to other notations such as Yourdon structure charts by means of forgetful functors. The framework also provides a theoretical basis for manipulating incomplete or unstructured specifications through refinement morphisms.

Since flow diagrams are used for problem analysis and communicating with users during early systems development, they are problem-oriented and are not necessarily structured. Some detection mechanism must be available for us to identify unstructuredness in the flow diagrams before we can convert them into structure charts or any other structured models. As a further illustration of the theoretical usefulness of our formal framework, we have derived a single criterion which is necessary and sufficient to identify unstructuredness in tasks. Namely, a connected task is unstructured if and only if there exist partially overlapping skeletons. As an illustration of the practical usefulness of our framework, we have developed a prototype system to implement the structured tasks. It enables users to draw a hierarchy of DeMarco data flow diagrams, review them to an appropriate level of detail, and zoom in/zoom out to lower/higher levels when required. It stores them internally as structured tasks, and transforms them automatically into Yourdon structure charts and Jackson structure texts.

This work originated from my Ph.D. research at the London School of Economics, University of London. I would like to express my sincere thanks to my supervisors Professor Ian Angell and Professor Ronald Stamper (now with the University of Twente) for their guidance throughout the doctoral programme. I am particularly indebted to Professor Joseph Goguen of the University of Oxford for his most encouraging comments and suggestions right from the very beginning of the project, and his endless advices on how to transform the thesis into the present work. I am grateful also to my external examiners Professor Bernie Cohen of the University of Surrey (now with Rex Thompson and Partners) and Professor John Campbell of University College London for their fair evaluation of the research. In addition, I will not forget the contributions of my former supervisor Professor Frank Land, now with the London Business School, without whose motivation this project would not have started in the first place.

Special thanks are due to Dr Haya Freedman of the Department of Mathematics, London School of Economics, for her knowledge base on initial algebra and category theory, to Mr Daniel Pong, now with the Bank of Montreal, Canada, for his technical assistance on requirements specification languages, and to Professor Francis Chin and other colleagues at the University of Hong Kong for their continuous interactions and support. My thanks should also go to Dr Peter Breuer and Dr Paritosh Pandya of the University of Oxford, Professor Rod Burstall of the University of Edinburgh, Dr Derek Coleman of Hewlett-Packard Laboratories, Professor Joe Davis of Indiana University, Professor Dr Hartmut Ehrig of Technische Universitaet Berlin, Professor Jim Emery of the University of Pennsylvania, Dr Kit Grindley of Price Waterhouse, Dr Don Sannella of the University of Bremen and Dr Sami Zahran of ICL Dataskil for their encouraging feedbacks.

The research would not have been successful without the study leaves granted by the University of Hong Kong, a Commonwealth Academic Staff Scholarship and a CICHE Visitorship at the University of London, a SERC Visiting Fellowship at the University of Oxford, a Fellowship of the International Conference on Information Systems, a Grant from the International Conference on Software Engineering, a Hong Kong and China Gas Co. Research Grant and a University of Hong Kong Research and Conference Grant.

Finally, I would like to mention a word of 謝謝 to my wife, children and other members of the family for their patience, love and care throughout the duration of the project.

CONTENTS

LIST OF TABLES AND FIGURES

1 Introduction

The specifier constructs a theory and attempts to refute it (hypothetico-deductive) while the knowledge engineer assembles a mass of empirical rules whose verisimilitude is unquestioned (empirico-deductive). The systems engineer, meanwhile, takes the utilitarian approach: if it works, use it.

—*Bernard Cohen et al.* (1986)

There are existing formalisms for description ... which are clear and well-understood, but lack the richness typical in descriptions which people find useful. They can serve as a universal basis for description but only in the same sense that a Turing machine can express any computation.

—*Terry Winograd* (1979)

Structured analysis and design methodologies have been recognized as the most popular tools in information systems development (Colter 1982). They are widely accepted by practising systems developers because of the top down nature of the methodologies and the graphical nature of the tools. A complex systems specification can be decomposed into a modular and hierarchical structure which is easily comprehensible. They enable practitioners to visualize the target systems and to communicate with users much more easily than conventional methods.

As a matter of fact, the structured methodologies have been designed by quite a number of distinct authors, each employing a number of models which vary in their graphical outlook. These include data flow diagrams (DeMarco 1978, Gane and Sarson 1979, McMenamin and Palmer 1984, Weinberg 1980), Jackson structure diagrams, Jackson structure texts (Jackson 1975), system specification diagrams, system implementation diagrams (Cameron 1986, Jackson 1983), Warnier/Orr diagrams (Orr 1977) and structure charts (Page-Jones 1988, Yourdon and Constantine 1979).

Different structured models have been found to be suitable for different situations depending on the characteristics of user requirements, the emphasis and the stage of development. In other words, we need more than one of these models during the development process of a typical system. If we provide practitioners with a computer-aided means of mapping one model to another, the efficiency of systems development can be greatly improved. Unfortunately, however, the models are only derived from the experience of the authors. In spite of the popularity of these models, relatively little work has been done in providing a theoretical framework for them. As a result, the transition from one model to another, although recommended by most authors, is arbitrary and only done manually. Automatic validation and development aids tend to be *ad hoc* and model-dependent.

On the other hand, many attempts have already been made to computerize the systems development environment. Some better known examples are ADS/SODA, EDDA, ISDOS, SAMM and SREM. Most of these approaches, however, are developed independently of existing structured analysis and design models. As pointed out in Davis (1982) and Martin (1983, 1984), practitioners are rather hesitant to use such new tools because they involve an unfamiliar formal language.

To solve the problem, we should not be designing yet another formal language from scratch. Instead, we must recognize the popularity of existing methodologies and apply mathematical theory to support them. In this book, we propose a unifying framework behind the structured models, approaching the problem from the viewpoints of initial algebra and category theory. We hope it will provide further insight for software engineers into systems development methodologies, guidelines for implementors of advanced CASE tools, and will open up a range of applications and problems for theoretical computer scientists.

In Chapter 2 of the book, we review the desirable features of a systems development environment. In Chapter 3, we examine the features of five related projects, thus comparing the structured models with other tools which have a better formal foundation but are less popular. In the initial algebra framework discussed in Chapter 4, we define a term algebra of structured systems, which can be mapped by unique homomorphisms to a DeMarco algebra of data flow diagrams, a Yourdon algebra of structure charts and a Jackson algebra of structure texts. In Chapter 5, we find that the proposed term algebra as well as the DeMarco, Yourdon and Jackson notations fit nicely into a category-theoretic framework. DeMarco data flow diagrams can be mapped to term algebras through free functors. Conversely, specifications in term algebras can be mapped to other notations such as Yourdon structure charts by means of functors. The framework also provides a theoretical basis for manipulating incomplete or unstructured specifications through refinement morphisms.

A further illustration of the theoretical usefulness of the concept of tasks is given in Chapter 6. Since flow diagrams are used for problem analysis and communicating with users during early systems development, they are problem-oriented and are not necessarily structured. Some detection mechanism must be available for us to identify any unstructuredness in the flow diagrams before we can convert them into structure charts or any other structured models. We prove that a single criterion is necessary and sufficient for identifying unstructured tasks. An illustration of the practical usefulness is given in Chapter 7, where we discuss a prototype system to implement the structured tasks. It enables users to draw a hierarchy of DeMarco data flow diagrams, review them to an appropriate level of detail, and zoom in/zoom out to lower/higher levels when required.

The project has been presented in Computer Science and Information Systems journals and conferences (see, for example, Tse 1986, 1987a, 1987b, 1987c, 1988). Feedbacks on our approach are favourable and encouraging.

2 Desirable Features of Systems Development Environments

2.1 INTRODUCTION

In the early days of stored program computers, the cost of software made up a mere 15 per cent of the total cost of information systems. But software cost has been escalating ever since, and is currently estimated to be over 80 per cent of the total (Boehm 1976), as illustrated in Figure 2.1. It is more alarming to note that more than two-thirds of the money is spent on the maintenance of existing software and only one-third on new developments.

In view of the escalation in software cost, research workers have been trying to improve on the languages used in systems specifications and to design development environments to support these languages. Quite a number of surveys on specification languages and their supporting environments have already been published. Some notable examples are Colter (1984), Jones (1979), Rock-Evans (1987), Wasserman

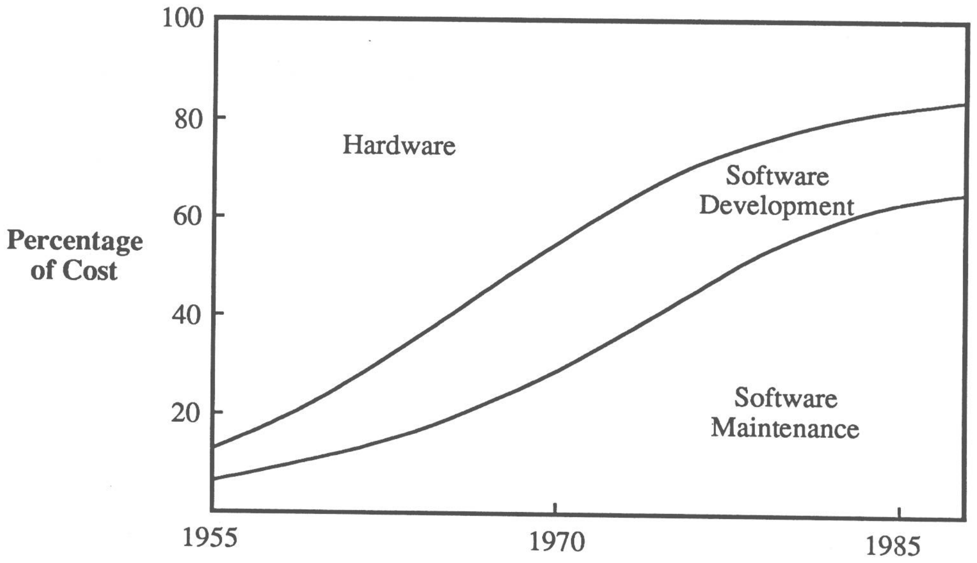

Figure 2.1 Hardware and Software Cost Trends

(1983) and Yau and Tsai (1986). Most of the authors (not excluding the present author (Tse and Pong 1982)) propose in the papers a list of desirable features of specification languages and/or development environments so as to provide a basis for judgement. One must confess that it is virtually impossible to invent yet another list of desirable features which would be better than those already proposed. Instead, we shall in this chapter consolidate the features already suggested by various authors and present them in the context of an engineering process. In the next chapter, we shall then use the proposed features as the basis to examine some of the established research in specification languages and their supporting environments.

Information systems development can be conceived as an engineering process. A graphical representation is shown in Figure 2.2. We must first of all build a model, which is a small-scaled abstract representation of the real world. All unnecessary details in the physical world which are irrelevant to the engineering process are removed. If the resulting model is still too complex, further abstractions may be necessary, until the problem is reduced to a manageable size. The model is then analysed and manipulated until a feasible solution is found. In engineering, diagrams and mathematics are often used because they have been found to be more suitable for manipulation than verbal descriptions. One representation may have to be transformed into another so that the most appropriate model for a given analysis can be used. When we solve an engineering problem, for instance, we may convert diagrams into equations or vice versa. Finally, if the abstract solution is accepted by the customer, a construction phase turns it into a real system.

A systems specification for the engineering process is important for several reasons:

(*a*) It serves as a communication medium between the user and the systems developer. It represents in a systematic fashion the current state of the real world, its problems and its future requirements.

(*b*) It enables the systems developer to turn real world problems into other forms which are more manageable in terms of size, complexity, human understanding and computer processability.

(*c*) It serves as the basis for the design, implementation, testing and maintenance of the target system.

In order for a systems specification to be useful for the entire engineering process, the specification language and its supporting environment must have the following features to cater for the respective stages:

2.2 ABSTRACTION OF THE REAL WORLD

A systems specification language is the medium for users to make a model of the real world and specify its problems and requirements. It is the bridge between a development environment and the users, including systems analysts, designers and end users. We must ensure that this interface is suitable to all concerned. The usual marketing

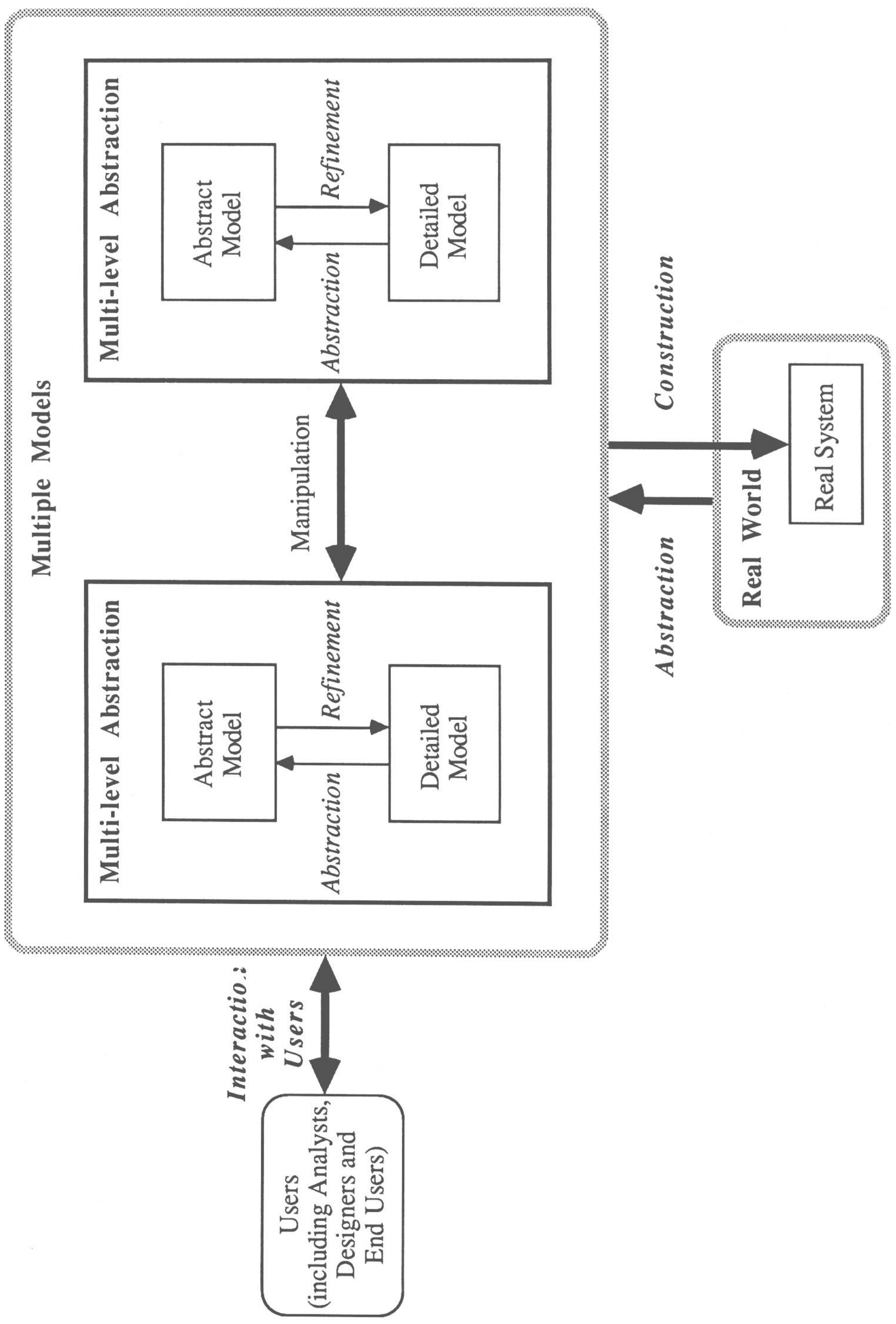

Figure 2.2 Schematic Concept of an Engineering Process

phrase ''user-friendliness'' is a bit too vague to act as a useful guide. Instead, we consider it essential for the systems specification language and its supporting environment to have the following properties:

2.2.1 User Familiarity of the Specification Language

It would be difficult for users to employ an entirely new specification language because of several reasons:

(*a*) There is an inertial effect from the point of view of users. They are not willing to try new methodologies which are not familiar, especially those involving formal languages.

(*b*) From the management point of view, a methodology which has been well-tested and which is being taught in various universities and polytechnics tends to be more acceptable than a newly proposed technique. It is easier to recruit staff members who are trained and experienced in an established method. It will be easier to maintain standards if the same methodology is used throughout the company. Managers in general find it safer to be old-fashioned than to try the latest innovation and regret afterwards.

When we propose a new systems development environment, therefore, we should not be inventing an entirely new language, with the hope that it will turn out to be the best in the world. Instead, we should try to use a currently available specification language which has most of the desirable features and, more importantly, has proven popularity among practitioners.

2.2.2 Language Style

To facilitate the automatic processing of a systems specification, it may be thought that the language used must be formal and mathematical in nature. A systems specification in a formal language, unfortunately, will be very difficult for users to understand. As pointed out in Davis (1982) and Martin (1983, 1984), practising systems analysts are very hesitant to such languages. Instead, the language must be easy to learn and easy to use. It must be close to the language employed by users in their respective domains of application. It must be precise and concise, or in other words, clear and to the point. Let us list out the possible classes of languages in order to arrive at some reasonable choice.

(*a*) *Textual language:* When we consider the use of a textual language for systems specification, we may like either a natural language or a more formal program-like language. There is little doubt that natural languages provide a better means of persuasion and more freedom of expression, especially in the early stages of systems development when a certain degree of uncertainty is involved. It is also more natural to the average end user and hence improves the user-understanding of a new situation. (The current book, for example, can only be written in a natural language, supplemented by graphics and other formal textual languages.)