An Intelligent Decision Support System for Debt Management

by

Alice M. Ireland

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
August 1990

ISBN 0-315-64442-7

DALHOUSIE UNIVERSITY

FACULTY OF GRADUATE STUDIES

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled " An Intelligent Decision Support System for Debt Management"

by Alice Marie Ireland

in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

©

Dated 23 July 1990.

External Examiner	Daniel R. Dolk
Research Supervisor	[illegible]
Examining Committee	[illegible]
	[illegible]
	[illegible]

DALHOUSIE UNIVERSITY

DATE 23 July 1990

AUTHOR Alice M. Ireland

TITLE An Intelligent Decision Support System

for Debt Management

Department or School School of Business Administration and Dept of Mathematics, Statistics and Computing Science

Degree Ph.D. Convocation October Year 1990

Alice M. Ireland
Signature of Author

For my parents,
who have taught me
to love learning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF LISP FILE LISTINGS

(microfiche inside back cover)

Abstract

In order to be used effectively by managers, decision support systems (DSS's) which use complex models require the assistance of human intermediaries in formulating, applying and managing models and in explaining and interpreting their output. Expert systems techniques offer ways of incorporating intermediary functions into DSS's so that they can be used directly by managers who are not modelling experts.

This dissertation reports on a project to integrate expert system and financial modelling techniques in a DSS for corporate debt planning. The system, called MIDAS (Manager's Intelligent Debt Advisory System), supports hierarchical planning in which a stochastic linear optimization model suggests a borrowing plan based on the user's problem description, the plan is modified using heuristics incorporated into the system, and stochastic simulation and (deterministic) cash flow projection are used to test the detailed implications of the resulting plan alternatives. The system is designed to assist the user in task selection and execution; model formulation, solution and modification; result analysis; and result explanation and interpretation.

MIDAS' design is based on five guiding principles: frame-based knowledge representation, object-oriented modelling, spreadsheet-oriented financial model structures, separation of knowledge and control or reasoning, and integration of models and heuristics through an underlying domain representation. A prototype system has been implemented which incorporates these principles for simulation, user assistance in task selection, and output management. Detailed design descriptions and documentation are presented in the dissertation. The research demonstrates the feasiblity of (a) hierarchical debt planning supported by multiple complex models and heuristics and (b) knowledge-based decision support for debt management. It also suggests a number of further research questions related to intelligent model management and explanation.

Acknowledgements

This research would not have been possible without the dedication, cooperation and support of many individuals and institutions.

I am grateful to the Faculty of Graduate Studies of Dalhousie University and its Dean, Kenneth T. Leffek, for the flexibility of the University's Interdisciplinary Studies program, under which I have been able to carry out this multidisciplinary effort. I very much appreciate the work of W.R.S. Sutherland and the members of my advisory committee in guiding me and themselves through a new and unfamiliar degree program. The financial assistance of the Social Sciences and Humanities Research Council of Canada and the Faculty of Graduate Studies enabled me to devote several years of full-time effort to this research.

I thank Harold Green, Don Keith and the staff of the Treasury Department of the Nova Scotia Power Corporation (NSPC) for their valuable input and continuinig enthusiasm for the MIDAS project. The financial and technical support of Unisys Corporation and NSPC's Management Information Systems Department for the project made it possible to use sophisticated hardware and software normally unavailable to graduate students, for which I feel extremely fortunate.

From the beginning, members of the project team contributed time, skills and exciting ideas; I particularly thank Hamilton Hinds for his help with our early explorations, Edwin Achorn of NSPC for sharing his expert systems experience and technical expertise, Gus Gassmann for his patience and sense of humour while we merged disparate views of the same problem, and Gordon Roberts for helpful discussions on financial management. The programming skills of Shailan Topiwala, Richard AuYeung, Yang Xiao Xen, Greg O'Malley and Zhao Yang helped to advance the development of the prototype system.

This project would not have been possible without the vision, guidance and energy of Michael A. H. Dempster, my advisor. Our many discussions have clarified project direction, unsnarled technical problems, helped me understand the wide range of theoretical material necessary for this work and given me insights into the meaning and process of research which will be invaluable throughout my academic career.

Producing the written dissertation was a monumental task, and my very special thanks go to Leslie Stockhausen for her typesetting skills, painstaking work and dedication to quality. Also, I greatly appreciate the help of the Dalhousie Computer Services staff at the School of Business Administration in supporting the complex combinations of hardware, software, communications and printers which produced charts and examples.

My special thanks go to my students and colleages at Dalhousie and at Saint Mary's University for their support and encouragement. Finally, my deepest love and appreciation go to my friends, especially Ramona, Rebecca and Sarah Roberts, for their understanding loyalty and faith in me.

1
Introduction

The ideal *decision support system* (DSS) provides integrated, multifunctional support to decision-makers in situations in which the decision process is not predefined or 'structured'. Its data and modelling components are integrated through a friendly user interface to form a user-directed, flexible environment within which the decision-maker can easily define many potential problem formulations and solutions and explore their implications.

Decision support systems for financial planning rely heavily on mathematical models for calculation and measurement of the profitability and risk of identified alternatives under uncertain future conditions. These systems, along with other model-based DSSs, would be most useful if they could provide flexible, easy-to-use facilities for selecting, setting up, running, modifying and interpreting any of a number of models relevant to the problem areas under consideration. In reality, however, financial DSSs often consist of single models, usually deterministic simulations, written in equation-oriented modelling languages or spreadsheets and databases linked through file transfers. More complex models, rarely used, are handled by human intermediaries who must set up, run and interpret the models before they are useful as decision support tools. The technology has not yet matched the vision.

A number of researchers have recognized the potential for expansion of DSS capabilities using expert systems techniques (Turban and Watkins 1986; Elam and Konsynski 1987; Kowalik 1987). By capturing human knowledge and reasoning, these techniques allow us to begin to incorporate a greater degree of human expertise in DSSs. In complex modelling situations, we should be able to begin to model the intermediary who stands between the user and the DSS, allowing the domain professional but system and model novice user to directly model problem situations and interpret the results.

This dissertation reports on a project exploring these possibilities for a domain-specific financial decision support system. The resulting system design supports direct operation by a novice user of deterministic and probabilistic optimization and simulation models. The proposed system reduces and in some cases eliminates the need for a human intermediary in complex financial modelling by combining symbolic and numeric processing to integrate domain and modelling knowledge, configure and manipulate models, integrate multiple complex models and rule-based reasoning, manage large volumes of output data, assist the user in deciding the sequence of task analysis steps, perform parametric and key factor impact analyses and interpret model results. Major sections of the proposed system have been prototyped, showing the feasibility of the proposed design. The research is important because it demonstrates integrated knowledge- and model-based debt management decision support, illustrates the complementary roles played by domain knowledge, models and heuristic knowledge in a system of this type, provides specific solutions to key model management issues in this domain and gives us some insights into general requirements and design principles for intelligent decision support systems.

1. Methodology

The research was carried out as a case study in which a design and proof-of-concept prototype system were developed for a specific decision situation within a cooperating corporation ('the Corporation'). The project followed the major steps in the generic systems development life cycle (Whitten *et al.* 1989, p. 89) modified to allow for (a) modular, evolutionary development of the system, beginning with the simplest and most user-demanded features, and (b) the use of prototyping for detailed design and system construction. These modifications were made to allow for the unexplored nature of the problem and of the development tool's capabilities as applied to problems of the type being studied.

Project steps were as follows:

1. *Scope and feasibility survey.* Initial discussions with Corporation staff and Dalhousie faculty led to a high-level problem description, project scoping and determination that the problem could be investigated on the Explorer/KEE hardware and software available in the Dalhousie School of Business Administration. Results of this step were presented in a project proposal (Ireland 1987).
2. *Analysis of current system.* The present system of debt planning was described and analyzed following detailed discussions with Corporation staff and their financial advisors. Results are presented as part of Chapter 3 of this dissertation.
3. *Requirements definition.* Detailed discussions with Corporation staff, their financial advisors and Dalhousie modelling experts led to development of a detailed problem description and functional requirements for the desired DSS, presented in Chapter 4.
4. *Identification of development priorities.* In order to narrow the scope of the prototype to fit time and resource limits, priorities for system development were identified based on Corporation needs, availability of expertise and apparent ease of implementation. These priorities are listed in Chapter 4.
5. *Iterative system prototyping.* In an experimental mode, prototypes of system modules were developed, tested and modified based on feedback from users and advisors and on experience with the development tools. The sequence of prototype development and experience gained is described in Chapter 9.
6. *Analysis of experience and results.* Based on our development experience and feedback from Corporation staff, key design and performance factors were identified and described; these form the basis for some suggested design principles for complex model-based DSSs for novice users discussed in Chapter 4. The appropriateness of the hardware and software used for the project was analyzed and is discussed in Chapter 9.

2. Problem domain

The problem domain under consideration is corporate debt planning for the Corporation, a Canadian Crown-owned electric utility. The Corporation is entirely debt-financed, so improved borrowing decisions will potentially result in significant dollar savings to the Corporation and taxpayers. The research system was developed to support the formation of long-, medium- and short-range borrowing plans and to assist in making individual borrowing decisions on short notice. The primary users of the system are the Corporation's Treasurer and Treasury staff, who are not trained in financial modelling and who have not previously undertaken extensive model-based analysis prior to making borrowing decisions.

3. Research contributions

Overall, this research demonstrates the feasibility of using expert systems representation and reasoning techniques to add human expertise to a complex model-based DSS. Its specific contributions are:

1. Articulation of a hierarchical approach to corporate debt planning applying stochastic programming, heuristic reasoning, simulation modelling and cash flow projection in a complementary manner.
2. Development of detailed designs for integrated frame- and rule-based modelling and system support, including task selection; model formulation, solution and modificaticn; and output data management.
3. Development of a technique for integration of multiple model types and heuristic reasoning through a common, object-oriented representation of domain objects.
4. Development of a design approach for user modelling assistance, including parametric and key factor impact analysis and explanation of the results of cash flow projections and financial simulations.
5. Articulation of design principles for dynamic stochastic portfolio management decision support systems for users who are not modelling experts.

4. Chapter organization

The body of this dissertation begins in Chapter 2 with a review of relevant literature in decision support systems, expert systems and model management. Chapter 3 describes the problem domain in detail and presents analytical approaches to debt management. Chapter 4 gives an overview of the system architecture and is followed by model specifications and detailed descriptions of major system components in Chapters 5 through 8. Chapter 9 presents a system assessment, including development experiences, performance characteristics and user feedback; Chapter 10 gives a sample planning session using the system. We close in Chapter 11 with a summary of conclusions and insights gained from the work, as well as suggestions for extending the research.

2
Background and Literature Review

This project addresses several issues in the application of *artificial intelligence* (*AI*) to modelling and model management in decision support systems. In this chapter we review the relevant DSS and AI literature, beginning with an overview of the performance goals and current status of modelling and decision support systems in general and in financial planning. We continue with a brief description of AI techniques and their potential for application to DSS and close with reviews of the needs for and research to date on the specific functions considered in this research. A detailed description of the problem domain and appropriate modelling techniques for the project is deferred until Chapter 3.

1. Decision support system concepts and goals

Decision support systems were first identified as a distinct class of computerized systems by Scott Morton under the name 'management decision systems' (Scott Morton 1971). He described them as 'interactive computer-based systems which help decision makers utilize data and models to solve unstructured problems'. Today a universally accepted definition of DSSs does not exist, but they are generally agreed to have the following goals and characteristics (Keen and Scott Morton 1978):

- incorporation of data and models
- assistance in decision-making in semistructured tasks
- support for, rather than replacement of, managerial judgment
- improvement of the effectiveness of the decision process, rather than its efficiency

- operation under manager control, without automating the decision process or imposing solutions.

In his classic paper, Sprague (1980) lists performance objectives for the ideal DSS as:

- support for making semistructured and unstructured decisions
- support for managers at all levels
- support for interdependent as well as independent decisions
- support for a variety of decision making processes
- support for all phases (intelligence, design and choice) of the decision process as described by Simon (1977)
- ease of use.

However, no one system has all these characteristics, and the performance criteria for a given system depend wholly on the task, organizational environment and decision maker(s) involved.

Sprague also describes the DSS architecture that has come to be generally accepted. A DSS consists of the following components:

- a *data subsystem* for storing, managing and reporting data and data relationships
- a *modelling subsystem* for creating and managing models that are linked to the database and used for analyzing problem situations
- a *dialogue management subsystem* for linking the user in a flexible and usable manner to the other subsystems.

In practice, the design, functions and relative importance of the data and modelling components in a particular DSS vary with the requirements of the system.

Alter (1980) categorizes DSSs as *data based* or *model based*, depending on the relative degree to which the system supports data analysis or modelling. The top priority for model based DSSs is to provide an accessible, flexible, user-controlled modelling environment, linked to databases to smoothly handle modelling and supporting data retrieval.

2. Use and limitations of financial models

Financial DSSs described in the literature are usually model-based. Financial management decisions are often, although not exclusively, concerned with acquiring or employing resources so as to maximize wealth; examples include capital budgeting, capital structure decisions, security valuation, lease versus buy decisions, bond refunding, cash budgeting and credit analysis, investment portfolio management, capital debt management and tax management (Heymann and Bloom 1988, pp. 108 and 118). An extremely common application is formulation of short- and long-term operating plans (McInnes and Carleton 1982). Financial models are used in these DSSs to calculate the profit, risk or financial statement implications of proposed problem solutions or to suggest optimal problem solutions based on such measurements.

A review by Shim and McGlade (1984) summarizes a number of surveys on the use of financial planning models in the U.S. and U.K. The use of financial models began with large-scale simulations in major corporations in the early 1960's (Naylor 1983); by 1982, 86% of Fortune 500 companies were using some type of financial planning model (Klein 1982). (The subsequent impact of microcomputers has undoubtedly expanded model use even further.) The most common applications found at surveyed companies were financial forecasting/planning and pro forma financial statement preparation; models were useful for testing decision alternatives, 'what-if' analysis, sensitivity analysis, analyzing best/worst case scenarios, goal seeking, optimization, simulation and report preparation. Financial models are commonly categorized as simulation or optimization models and as probabilistic or deterministic (Hammond 1974, Power 1975, Grinyer and Wooller 1975, McInnes and Carleton 1982). Although many types of models can be applied to financial problems, most corporate models found in the surveys were deterministic simulations (cash flow or profit projections). Probabilistic considerations were generally not incorporated into models and optimization was rarely used (Naylor 1983, McInnes and Carleton 1982).

Model use in DSS has presented many commonly recognized difficulties. These arise from two sources: organizational factors associated with model development and implementation and technical limitations of traditional modelling techniques. Examples of organizational factors are lack of the manager's involvement in model development (Power 1975, Naylor 1976, Klein 1982) and his/her fear of loss of control over decisions (McInnes and Carleton 1982); these are addressed in part through a participative, evolutionary DSS development process (Sprague 1980). Technical factors include:

- model limitations: simplification of reality, lack of flexibility, inability to quantify important variables or relationships, inability to cope with discontinuities; poor documentation (Naylor 1976, Brennan and Elam 1986*a*, Heymann and Bloom 1988)
- input limitations: data unavailability, excessive requirements, inability to explicitly articulate goals and goal weightings (Naylor 1976, McInnes and Carleton 1982, Brennan and Elam 1986*a*)
- output and interpretation limitations: difficulty interpreting probability distribution output from simulations; inability to communicate results in a way most likely to be understood by the user or to explain why results are obtained as they are (Brennan and Elam 1986*a*, Bryant 1987).

DSS design goals directly address these technical difficulties. DSSs are intended to make models directly available to and controlled by managerial users (Keen 1980). Ideally, a DSS uses multiple models so that deficiencies in one are compensated for by strengths in others. Input is obtained directly from the DSS database, reducing onerous user input activities. An easy-to-use interface increases the user's control over analysis and reporting. And results are to be reported in a choice of formats that are easier for managers to interpret.

However, significant gaps exist between the ideal DSS and the reality. Empirical evidence suggests that although DSSs do provide a more congenial modelling environment, they are often not used directly by managers, but are instead operated by staff intermediaries who carry out analyses and present the results to managers for use in 'off line' decision making (Keen 1976, Andriole 1982, Hogue and Watson 1984). Some researchers believe that a trained intermediary who understands the capabilities and limitations of a specific DSS is necessary for its successful use (Keen 1976, Alter 1980). However, Elam and Konsynski (1987) argue that the use of intermediaries limits the benefits of DSS use by reducing the manager's control over the decision making process; limiting the manager's involvement in and commitment to modelling; encouraging overdependence on modelling and insufficient attention to exploration, interpretation of results and implementation; and restricting DSS use to companies with the resources to support specialized technical staff. In their view, the full potential of DSSs cannot be demonstrated until the need for intermediaries is eliminated.

Because many are proprietary, detailed descriptions of innovative financial models and financial DSSs are not widely found in the literature. We therefore have little direct evidence of the extent to which companies are overcoming the limitations of financial modelling and DSSs identified above. However, there is overwhelming evidence in journals and the popular media of the popularity of microcomputer-based spreadsheet software, which principally allows users to directly develop deterministic projections through a worksheet interface; the other major type of financial DSS tool in common use is the financial modelling language, which allows more complex simulations to be built using English-like statements (Turban 1988, pp. 183–186). From this evidence we can guess that the prevalence of deterministic projections in financial modelling has not changed significantly since the most recent reported surveys. Spreadsheets and modelling languages provide flexible user interfaces for model development, so the need for intermediaries may be reduced for these types of models; however, optimization and probabilistic models appear to be poorly accepted and to need human assistance if and when they are used. Financial DSSs, therefore, must incorporate intermediary functions if more powerful models are to be accepted as standard management tools.

3. Overview of artificial intelligence and expert systems

Because they require human expertise for a variety of specific tasks, intermediary functions have been suggested as promising applications of artificial intelligence techniques in DSS (Turban and Watkins 1986, Henderson 1987, Elam and Konsynski 1987).

Artificial intelligence involves the study of human thought processes and their emulation by machine (Turban 1988, p. 312). *Expert systems* (ES) is a branch of applied AI in which computer programs use the knowledge and reasoning processes of human experts to solve specific types of problems (Turban 1988, p. 321); expert systems techniques appear particularly useful for building more 'human' functionality into decision support systems (Turban and Watkins 1986). The following overview of expert systems is based largely on Turban (1988) and Barr and Feigenbaum (1981).

Current expert systems can solve limited-scope problems and provide simple explanations. Examples of large-scale commercial expert system successes include XCON, which configures DEC VAX computer systems (Waterman 1986) and a system for handling unusual credit requests for American Express (Davis 1987).

Expert sytems differ from conventional programs in several ways. First, they use symbolic rather than numeric processing. In common with other AI systems, they store and manipulate lists of characters which may be numbers but also may be words, names, or sentences expressing concrete or abstract ideas. Based on the Newell-Simon model of human information processing (Newell and Simon 1972), symbolic processing allows AI systems to reproduce human reasoning such as, 'If a company's liabilities are greater in value than its assets, then it is bankrupt.'

Second, expert systems and other AI systems rely on heuristic rather than algorithmic reasoning. Human experts routinely use *heuristics*, or 'rules of thumb', to solve problems with which they are familiar; heuristic reasoning applies these rules when they are useful in a situation, rather than on a predefined step-by-step algorithm or routine which is independent of the characteristics of the particular problem.

Third, expert systems separate knowledge from reasoning. *Knowledge* in an ES includes facts about the problem of domain, procedures to be followed given certain conditions, and rules about conclusions to be reached in certain circumstances; this knowledge is stored using any of several *knowledge representation* techniques. Generalized *reasoning* routines select appropriate knowledge and apply it to specific problems. The types of knowledge representation and reasoning relevant to this project are briefly described below.

3.1 Rules and rule-based reasoning

The most common form of knowledge representation in expert systems is that of *production rules*, developed as part of the Newell–Simon theory of cognition. A *rule* is an 'If-then' structure with condition and consequence components, e.g. 'If X is a full-time student, then X pays fees of $1800.' A rule is applied, or 'fired', in solving a problem when its condition is satisfied by the facts of the situation; in this example, the fact 'Janet is a full-time student' leads to the inference that 'Janet pays fees of $1800.' A rule's consequence can be a conclusion (fact), as in this example or an action (procedure), as in 'If X is a full-time student, then send X registration information.'

A rule-based expert system solves a problem by applying a number of rules in sequence; each rule modifies the problem's facts until a solution is found or until no more rules apply. *Rule-based inference* is done through either forward chaining or backward chaining. *Forward chaining*, the data driven approach, begins with a fact and applies all relevant rules to infer all possible conclusions; *backward chaining*, referred to as goal driven reasoning, begins with a desired goal (consequence) and searches through rules to attempt to support the goal through known or inferred facts. Complex expert systems control rule-based inference through rule subgroups, combined forward and backward chaining and *meta-rules*, which specify which groups of rules to apply at given times.

3.2 Semantic networks

A *semantic network* represents factual knowledge about a problem through nodes and links (Quillian 1968). Nodes and links are named; nodes represent objects or concepts, and links represent relationships between pairs of nodes. Nodes can be arranged in hierarchies of classes representing increasingly specialized object types. The HAS-A link attaches facts to objects in the network, e.g. Bird HAS Feathers; the IS-A link relates specific objects to classes of objects, e.g. Bismarck IS-A Cat.

IS-A links establish *inheritance* for semantic networks, whereby nodes representing subclasses or individuals acquire the properties of their parent classes. Inheritance allows classification of unknown objects based on their observed properties and subsequence inference of previously unknown object properties based on class characteristics.

3.3. Frames and object-oriented programming

Frames are an extension of semantic networks in which nodes are rich data structures using both *declarative* (factual) and *procedural* knowledge to represent potentially complex objects or behaviours (Minsky 1975). A frame is a data structure representing a prototypical object, concept or situation; each frame consists of a group of named *slots* or attributes. Each slot, in turn, has a name, value and several *facets* which may include value constraints, default values, *demons* (procedures or rules that fire automatically if the value is accessed or changed), procedures for calculating slot values or other knowledge about the slot (Fikes and Kehler 1985).

Frames represent object classes and *instances* (individual class members) through inheritance networks. Objects are described as successively more specialized classes which inherit the properties of their parent classes unless specifically overridden. Instances inherit all the properties of their parent classes and also inherit default values for attributes that are not specifically known. Reasoning with frames takes place either through inheritance or expectation-driven processing, in which slot values are filled in to classify objects or situations and confirm expectations about them. Frames are useful for modelling (see below) and for organizing knowledge in complex problem domains; for example, CENTAUR (Aikens 1983) uses frames to explicitly represent control knowledge for prototypical situations to be diagnosed and to focus the application of production rule sets.

Object-oriented programming (OOP) is an extension of frame representation which uses frames to combine data and procedures in *objects* for modular system organization and implementation (Stefik and Bobrow 1986, Tello 1989). Objects have attribute slots which contain descriptive values and *method* slots which specify behaviours or functions the object can perform on request. Like frames, objects are defined in inheritance networks which pass both attributes and methods to child subclasses and instances. In strict OOP, objects are entirely self-contained and their data can only be accessed by their own methods. OOP originated with Xerox research on graphic interfaces and provides techniques for building models and other software using reusable, modifiable classes of program code.

Access-oriented programming is an extension of frames in which accessing or changing data in slots causes demons to fire, performing procedures (Stefik *et al.* 1986). It is often used for simulations in which behaviours are automatically performed based on monitored conditions. Object-oriented and access-oriented programming are useful tools for *model-based expert systems*, which allow reasoning from first principles about problems for which expert heuristics are not known.

Worlds (Filman 1988) represent sets of related facts such as the assumptions and conditions that characterize a solution alternative for a planning problem. In an object-oriented system, a world is essentially a copy of the knowledge base in which all objects and attribute values are inherited unless overridden; it thus represents a unique problem state which may be a variation of the original state. Worlds are often used in conjunction with a *truth maintenance system* (TMS) which maintains the chains of reasoning which justify facts found to be true in each world under consideration. Worlds are useful for incremental construction of solutions for configuration problems, hypothetical reasoning, exploring the implications of multiple scenarios and reasoning with alternate sets of beliefs.

3.4 Problems addressed by expert systems

Expert systems are currently successful at solving complex problems in narrow domains, often in situations in which either the required volume of knowledge exceeds the scope of one individual or available time is too short for a person to effectively solve the problem. Common ES task types include interpretation, diagnosis, monitoring, prediction, planning and design. Many model management and intermediary functions fall into these categories; examples include diagnosing the type of model to use in a given situation (Binbasioglu and Jarke 1986, 1987), designing a problem-solving strategy (Sivasankaran and Jarke 1985), formulating a linear programming model (Murphy and Stohr 1986, Lee *et al.* 1989) and interpreting model output (Bouwman 1983; Kosy and Wise 1986; Wise and Kosy 1986).

4. Expert systems applications to intermediary tasks

The intermediary tasks which appear to lend themselves to implementation through expert systems techniques can be broadly classified as either model management tasks (Sprague and Carlson 1982) or user-model interaction tasks (Elam and Konsynski 1987).

Model management tasks are those which allow users to build, store, access and use models within a computerized system, and to integrate multiple models where problems cannot be fully analyzed using single models. Sprague and Carlson (1982) list the following model management functions as key for a successful DSS:

- the ability to create new models quickly and easily
- the ability to access and integrate model 'building blocks'
- the ability to interrelate these models with appropriate linkages through the data base
- the ability to catalogue and maintain a wide range of models, supporting all levels of users
- the ability to manage the model base with functions analogous to data base management (i.e. mechanisms for storing, cataloguing, linking and accessing models).

The first three of these functions allow model formulation, modification and integration regardless of the size and diversity of the 'model base' or its users; the importance of the last two functions increases with the number of models to be managed and the degree to which the DSS is a general support facility for many members of an organization. All functions deal with technical aspects of model building, storage, manipulation and solution within computerized systems rather than with the application of models to problems.

User-model interaction tasks are those which are required in order to effectively apply models to real problems. These tasks, identified by Elam and Konsynski (1987), are those which establish and describe relationships among problems, models and model results; examples are (a) deciding when and how to use particular models to analyze problems, (b) evaluating model results to see how closely they correspond to expectations, and (c) interpreting model results in problem-domain terms. Human intermediaries accomplish these tasks using modelling knowledge and *meta-modelling knowledge*; that is, knowledge about how to formulate, use or interpret the models being used (Bonczek *et al.* 1984). A full list of user-model interaction tasks is given in Figure 2.1.

This dissertation research focuses on two groups of tasks from the above categories: (a) model formulation, synthesis and integration, and (b) result analysis, interpretation and causal explanation. These tasks were selected because they were especially important to successful system use in the case study situation. The following sections of this chapter review current research on these tasks.

5. Model formulation

Model formulation involves both model *specification* (definition of the objective function (if applicable), variables, coefficients and constraints and other mathematical relationships for a given problem) and *instantiation* (assignment of numerical values for the precise problem being analyzed). As discussed in Chapter 3, the problem domain and decision process for this research are limited to hierarchical planning for long-term debt management using two types of mathematical modelling (optimization and simulation). Each model type is highly complex overall; moreover, both models must often be *respecified* and *re-instantiated* in response to changing problem requirements and assumptions. Model formulation for this system must therefore provide great flexibility in handling complex models; it must also provide extensive novice-user assistance because the intended system users are not modelling experts.

5.1 Modular model specification

Model *modularization* and *synthesis* from reusable submodels are widely regarded as promising means to achieve model flexibility and to leverage modelling efforts so that modelling does not have to be continually carried out 'from scratch' (Sprague and Carlson 1982, Henderson 1987, Elam and Konsynski 1987). Modularization has long been used in the form of model subroutines in FORTRAN and other procedural languages and is now common as the basis for higher-level user-controlled modelling. Page-Jones (1988) lists the following guidelines for system modularization: *top-down design*, *maximum module cohesion* (task singularity within modules)

Formulation tasks

Formulate	–	Formulate a new decision model if an appropriate one does not exist in the model base.
Explore	–	Explore ideas and analyze issues.
Choose	–	Choose an existing model from the model base.

Analysis tasks

Match	–	Identify and test (from a base of existing model structures) the applicability of a model and its associated solution approach to a particular problem.
Expect	–	Detect, explain and suggest solutions for abnormal behaviour based on user-supplied expectations about model behavior and/or the history of the model's utility.
Plan	–	Determine ways to perform analyses to reach predetermined goals.
Cause	–	Identify causal relationships between model entities.
Recommend	–	Identify, evaluate and choose among potential courses of action.
Synthesize	–	Synthesize new models from model fragments that prove locally successful.

Interpretation tasks

Explain	–	Generate explanatory models that provide intuitively reasonable explanations for the model's results.
Interpret	–	Interpret the analysis solution or results in the context of the problem semantics.
Present	–	Select appropriate report formats for requested information.

Figure 2.1 User-model interaction tasks: Opportunities for AI application. *Source:* Elam and Konsynski (1987).

and *minimum coupling* (data and functional dependence between modules), adapted from DeMarco (1978).

The four main design issues which must be solved in model modularization and synthesis are *module definition* (including cohesion), *module selection* where a module library already exists, *communication* (coupling) and *control*. Models requiring sequential calculations are likely to define modules according to calculation task. One example is the ACS system (Sivasankaran and Jarke 1985), which represents individual actuarial formulas in a knowledge-based system for building and executing *actuarial subroutines* in response to a user's problem statement. The WHIMS system (Miller and Katz 1986) for *discrete event simulation* model building relies on the user to define modules in the form of procedural code for simulation calculations.

Because linear optimization models are defined by integrated matrices, they cannot be decomposed into sequentially executed subroutines. *Linear program* (LP) *model building* is therefore a complex process involving specification and linking of matrix 'building blocks' expressing processes and constraints in the underlying problem (Murphy and Stohr 1986); formulation from an underlying representation of domain entities (Binbasioglu and Jarke 1986, 1987); or formulation from a combination of structural and domain representations (Lee *et al.* 1989). The resulting matrix is subsequently solved by an external solver program.

Systems which build models by selecting relevant modules from a submodule library use a variety of selection criteria which reflect the design goals of the systems. Examples include execution cost (Sivasankaran and Jarke 1985) and mixed qualitative and quantitative constraints (Dhar and Pople 1987, Dhar and Croker 1988). In the LP formulators previously cited, problem descriptions in a knowledge base are linked to equation specifications or matrix blocks; identification or construction of the relevant problem description triggers selection of the appropriate mathematical components.

For sequentially executed modules, communications must be specified so that correct data values and control parameters are passed from one step to the next. Communications usually take place either through common memory or database access; links are usually defined through consistent input/output or attribute names, either user-defined as in WHIMS or within the system's module descriptions as in ACS or Dhar and Pople's PLANET. Binbasioglu and Jarke's system links LP models in sequence through common decision variables.

Module control requires mechanisms for choosing a feasible order and for calling modules in the chosen order. Execution order is user-specified in WHIMS, and control is handled through evaluation of templates. ACS chains calculations to produce the least-cost derivation of desired output; solution plans are represented as lists of formulas which are evaluated in sequence. More complex control mechanisms utilize AI inference techniques such as *forward* and *backward chaining*, connection graphs, demons and scripts (Fedorowicz and Williams 1986, Dhar and Pople 1987).

5.2 Model instantiation

Model instantiation is the task of translating the model specification for a problem into a complete model with all required data values. In the above systems, coefficient and parameter values are retrieved through file reads (WHIMS) or database access (ACS); where files or databases are used, external links or read mechanisms must be established. Where object-oriented representations are used, necessary data is either *encapsulated* within the knowledge base as object attribute values (Binbasioglu and Jarke 1986, 1987; Dhar and Croker 1988) or *extracted* from a separate database using procedural code in methods or demons (Lee *et al.* 1989).

5.3 Current system limitations

The model formulation systems cited above demonstrate the power of modularization for specific domains and problem types; however, each is limited in certain ways. For example, the simulations referred to all appear to be *deterministic*; at least, no mention is made of stochastic simulations in the reports. They therefore require less user-assistance expertise than do models which require sampling from random-variable distributions.

Also, each system handles only a single model type of limited complexity. WHIMS and ACS handle sequentially-organized formulas or simulation modules, the linear program formulators cited construct only this type of model, and PLANET and its extensions model production simulations. None addresses the issue of consistency of assumptions, internal relationships and objective functions across *multiple models.*

As mentioned earlier, one goal of this research is to extend model formulation capabilities beyond the limitations just described. Specifically, our goal is to consistently and flexibly formulate (a) larger and more complex stochastic models and (b) multiple model types in a single system while retaining the user-responsiveness provided by these single-model systems.

6. Integration of multiple models and heuristic reasoning

DSS researchers have long recognized that management problems can rarely be completely solved or even completely analyzed with a single model. For financial management we have available many models with different strengths and weaknesses; the categorization into *optimization* and *simulation* model types gives us two broad classes of tools.

Optimization models are tools for quickly considering all alternative solutions to a circumscribed problem and identifying the optimal choice, in terms of a mathematically defined objective; as documented earlier in this chapter, their complexity, simplification of reality and 'black box' approach have limited their use among financial managers even though they attempt to provide strong support for rational decision-making through consideration of all factors. (Lane and Hutchinson (1980) provide an interesting illustration of these difficulties for a bank portfolio management model; in this case, managers reacted negatively to 'prescriptive' decision variable values but readily used the cost information in dual variable values to guide investment decisions.)

Stochastic simulations and deterministic projection models complement optimization models; they can capture the interactions and relationships of a situation in detail and in terms users can understand, and they support 'satisficing' (Simon 1977) or finding an adequate solution rather than the best one. As we have seen, deterministic projections are by far the most commonly used corporate financial models.

Because DSSs are intended to provide a rich, flexible problem-solving environment, support for and integration of multiple model types is an important DSS function. Multiple-model integration extends model synthesis considerations to larger cooperating 'modules' in the form of full-scale models with varying objectives and solution techniques. *Consistency* and *communication* are key design goals. Like smaller model fragments, well-integrated models must be consistent in their assumptions, parameter definitions and mathematical relationships and must remain consistent when modified in response to changing problem requirements. Each model must also be able to communicate its results to other models and to use their results as appropriate.

Generalized techniques for integration of multiple models have only begun to be addressed by researchers; a significant effort in this direction is the structured modelling approach taken by Geoffrion (1987), Dolk (1990) and Dolk and Kottemann (1990). However, working examples of multiple model integration are found in the DSS literature. PROJECTOR (Meador and Ness 1974), for example, provides multiple regression, exponential smoothing, goal programming and optimization linked through a common database. Another approach uses forecasting or statistical techniques to generate parameters for optimization or simulation models (Dyer and Mulvey 1983, Turban 1988). In a comprehensive financial planning system, Hamilton and Moses (1973, 1974) integrate mixed-integer programming, deterministic projection, econometric forecasting routines and risk analysis linked through a set of eight databases maintained through an information management subsystem.

Model consistency and communication (for example, ensuring that one model's output meets another model's input requirements) are left to the user in these systems. Users must therefore become familiar with modelling techniques and assumptions or must use intermediaries, as we have seen previously.

Even with the use of several mathematical models, most management problems cannot be fully captured and analyzed. Qualitative, judgmental and political factors are not neatly expressed in mathematical algorithms yet are critical to successful solution of management problems.

Because they usually encode rules and carry out heuristic reasoning, expert systems techniques have been suggested as ways to augment models for more comprehensive problem analysis. Turban and Watkins (1986) suggest the use of expert systems as separate components in a DSS, sharing in or independently handling specific steps in the decision process. One example is Lee's *Post-Model Analysis Approach* (Lee and Kang 1988, Lee and Lee 1987, Lee and Hurst 1988, Lee 1989, Lee and Kim 1989), which uses rule-based reasoning to modify linear program results to meet qualitative objectives. Other examples of such expert systems components are found in a system for *strategy formulation* (Meador, Keen and Guyote 1984) and the PLANPOWER expert system for *financial planning*, which combines projection with rule-based reasoning in producing personal financial plans (Stansfield and Greenfeld 1987).

This research extends these results to demonstrate the integration of stochastic as well as deterministic optimization modelling, simulation modelling, projection and rule-based reasoning with consistency maintained by the system rather than the user, allowing a novice user to operate all parts of the system without detailed knowledge of their technical requirements.

7. Analysis of model results

Parametric analysis for optimization models and sensitivity and key factor impact analysis for simulations and projections require knowledge about identifying key assumptions and appropriately modifying and re-executing models. Although these analyses are often user-constrained, automatic parametric analysis (solving a model with a series of values for a key constraint right-hand side) is an accepted feature of many optimization systems (Turban 1988). An interesting example of sensitivity analysis assistance in a DSS is found in a generalized business modelling system, which not only varies parameter values on request but revises a plan automatically when the variation causes a constraint to be violated (Dhar and Croker 1988).

8. Model explanation and output interpretation

Research cited earlier in this chapter suggests that a model is more likely to influence decisions if the decision maker understands the significance and limitations of model output in domain terms. Two of the major tasks facing human intermediaries are therefore (a) to explain the concepts, calculations, causal relationship and limitations of models in terms understood by decision makers, and (b) to interpret model output by translating results into the semantics of the domain (Brennan and Elam 1986*b*).

Explanation and interpretation of financial models have been approached in two ways in recent research, as described below: through *qualitative* (*symbolic*) *reasoning* and through *quantitative calculation tracing* using representation of the specific equations comprising a simple model.

Explanation of model relationships and results for simple financial models in the form of financial statements is demonstrated by Bouwman (1983). This system reasons qualitatively with verbal descriptions of changes in financial figures over time; its main goal is to develop a causal description for a company's present condition. The ROME system (Kosy and Wise 1986; Wise and Kosy 1986) explains resource allocation plans quantitatively, using underlying mathematical equations describing known relationships in the plans; their system successfully identifies the most important causal factors in a result and develops *causal path descriptions* for chains of simple calculations.

Both systems concentrate on relatively simple models in which relationships can be causally traced in sequence. For the more complex models considered in this research, the detailed tracing used in ROME may not be practical and a *higher-level* description is needed which highlights key points and relationships. Brennan and Elam (1986) point out the need to cut through excessive *output* detail in presenting model results to managers; Bouwman's approach provides a useful starting point in that it 'chunks' together complex relationships underlying financial reports and uses the chunks as the basis for diagnosing and describing report changes.

Interpretation and explanation of more complex models has been addressed by Greenberg (1987*a*,*b*, 1988; Greenberg and Lundgren 1989; Greenberg and Murphy 1989) in the ANALYZE system, which provides qualitative explanations of deterministic linear program output for energy models. This system analyses an output matrix by first referring to a syntax file of templates for translating row (constraint) and column (activity) names into natural language; a semantic model then uses rules to produce full descriptions of rows or columns and, in addition, analyzes patterns within the matrix to identify infeasibility conditions.

Several proprietary expert systems explain rule- and model-generated financial results for the purposes of financial planning or strategic planning. One publicized example is PLANPOWER (Stansfield and Greenfeld 1987), although the design principles and details used in this system are not available. Another example is PMS (Lee and Kim 1989), which uses 'because' clauses in rules to generate explanations for its rule-based modification of input and output for a quadratic programming portfolio model.

9. The role of domain knowledge

One of the issues being addressed in this project is the degree to which domain or problem knowledge can provide an underlying integrating structure for multiple models and other forms of reasoning about a single problem. AI researchers have realized the importance of specific domain knowledge in providing the 'expertise', manipulated with a generalized reasoning facility, for expert systems (Davis 1984). Similarly, the inclusion of domain knowledge in a DSS allows the system to 'understand' problem descriptions and map between them and appropriate models, results or explanations. Examples of domain-specific DSSs include ACS, an *actuarial modelling system* which is able to formulate and execute problem-solving plans using multiple models sequenced according to varying problem descriptions (Sivasankaran and Jarke 1985); a *production planning system* which formulates linear programs based on object-oriented domain entity descriptions (Binbasioglu and Jarke 1986); UNIK-OPT, which uses frame representations of domain and model structure knowledge to *formulate linear programming models* for *production planning* and *portfolio management* (Lee *et al.* 1989); and the ANALYZE system mentioned earlier, which explains large linear programming models for *energy policy formulation*.

This research investigates the use of domain knowledge for formulation and integration of multiple complex models—models more difficult than those so far found in the literature. Geoffrion's (1987) structured modelling approach is a generalized modelling specification framework which proposes the use of operators to create multiple models from a single object and relationship specification; this research takes a similar approach, limited to a single domain.

10. Summary

Financial models provide information for managerial decisions, but they require extensive modelling expertise to be effectively used. This conceptual and heuristic knowledge has traditionally been found either in the manager or in intermediaries who advise on model application, thus limiting the use of the techniques to situations where managers have advanced training or consultants are readily available. This suggests that modelling expertise is a useful application of expert systems, which provide knowledge representation and reasoning ability and serve to extend expertise to locations where it is in short supply. A review of current research on the specific modelling assistance tasks of formulation, synthesis, integration, result analysis, interpretation and causal explanation suggests that while significant progress has been made in providing these within DSSs, they have not yet been made available for multiple types of complex models.

This research applies expert systems techniques to model formulation, integration, heuristic extension, manipulation, explanation and interpretation in a domain-specific decision support system. In the next chapter we describe the problem domain and the types and extent of modelling support available for it.

3
Domain Description

Debt is one of the two main sources (the other is *equity*) of funding for business organizations in the capitalist economic system. Debts transfer funds from lenders to borrowers in exchange for *interest payments* at stated rates and times, repayment of *principal* (the amount borrowed) according to a specified schedule and other provisions as explained below.

The corporate debt manager is responsible for choosing the amounts, types and sources of debt utilized by the firm according to various corporate objectives. *Cost minimization* within an acceptable *risk level* is usually the top concern, consistent with profit maximization and stabilization for the firm (Brealey *et al.* 1986). Because cost and risk are determined by future economic conditions as reflected in interest rate movements, debt management is carried on in a highly uncertain environment and relies on expert judgment as well as analysis.

This chapter first surveys the debt management environment, including debt types and covenants, financial markets, cost and risk determinants and risk management techniques. It then summarizes the finance theorists' view of the problem and describes analytical approaches to assist in debt planning decisions. Finally, the borrowing process at the Corporation is outlined and the roles desired for a decision support system for the process identified.

The following descriptions of debt types, borrowing costs and risks are based on material in Fabozzi and Pollack (1983) and Hunter (1986) and on personal discussions with staff at the Toronto office of Scotia McLeod Limited.

1. Debt types and covenants

The corporate debt manager can choose from a number of types of debt which vary in their interest provisions, repayment terms, cost characteristics and restrictions or privileges for the borrowing corporation. In any debt transaction, both the borrower and lender face certain risks (variability of costs or returns); individual debt provisions often shift risks from lender to borrower or vice versa with a corresponding movement in expected cost or return. Borrowing risks will be discussed in more detail later in this chapter; major debt types used in Canada are described below.

1.1 Short-term debt

Short-term debt instruments are generally defined as loans which mature (are repayable) in one year or less. (This definition may vary, however; for example, the Scotia McLeod short-term bond index measures returns on bonds having one- to five-year maturities.) Short-term securities with low risk of default are traded in what is referred to as the money market; the two common corporate money market instruments are *commercial paper* (unsecured short-term notes from the most credit-worthy corporations) and *bankers' acceptances* (short-term notes guaranteed by banks). The money market has no formal organization but operates through dealers and brokers who specialize in particular types of instruments; its operation allows firms to adjust their liquidity positions by investing surplus cash and borrowing to cover short-term cash deficits. Commercial paper and bankers' acceptances are issued at a discount from face value, with interest effectively paid when the full face value is repaid at maturity.

Outside the money market, corporations also obtain short-term debt through *bank credit* and *trade (supplier) credit*; these credit arrangements are often secured by receivables or inventory and usually require periodic interest payments at either fixed or floating rates.

1.2 Intermediate and long-term debt

Intermediate-term debt generally includes debt with maturity of between one and ten years, although this definition, like that of short-term debt, varies. (The Scotia McLeod mid-term bond index, for example, measures returns for five to ten-year bonds.) *Long-term debt* is generally considered to include maturities greater than ten years.

Intermediate and long-term debt instruments are traded in capital markets and have numerous contractual forms. Publicly issued *corporate bonds* are initially sold in units of $1,000 (referred to as *par value*) by dealers acting as underwriters and are traded in secondary markets. Each bond issue is overseen by a trustee (bank or trust company) who supervises the initial sales and then acts as a watchdog for the bondholders to ensure that all provisions of the bond *indenture* (contract between issuer and investors) are carried out. Bonds are often secured by liens on all property of the issuer. Unsecured bonds are called *debentures*. Bonds are issued with *term* (time to maturity) of from one to thirty years. Most bond issues provide for semiannual or annual interest payments at a rate (the *coupon rate*) fixed in the bond indenture; the actual price at which each bond unit is sold is higher or lower than face value to adjust the bond's return to the investor to the current market rate at the sale date. *Zero-coupon bonds* (also known as *original issue discount bonds*) pay no interest and are issued at discounts reflecting the market interest rate at the time of issue; because accrued income on these bonds is taxable to holders even though no cash flow is generated, the market rate for zero-coupon bonds is higher than for a comparable coupon bond.

Corporate bond indentures include numerous provisions for repayment and refunding. Call and sinking fund provisions are most common. A *call provision* allows the borrower (bond issuer) to repay part or all of the outstanding amount at prespecified dates at stated premiums over par value. Call provisions sometimes state that bonds may not be called in order to refund at a lower interest rate, although they can be called for repayment from surplus cash. A call provision which does not restrict refunding lowers risk to the issuer, since it allows the issuer to take advantage of future lower interest rates; it increases risk to the lender since the bond contract may be terminated before maturity. Bonds with call provisions therefore generally have *higher* coupon rates than comparable noncallable bonds. Call provisions are often *deferred* so that a call cannot take place during the first five or ten years following the issue date.

A *redemption* or *retraction provision* allows early retirement, at stated dates and at stated discounts, of the bond at the option of the bondholder (lender). It enables the bondholder to reinvest should interest rates rise and results in lower cost but greater risk to the issuer.

A *sinking fund provision* requires the issuing corporation to either accumulate funds for repayment at maturity or to periodically redeem bonds before maturity. This reduces default risk and lowers the coupon rate.

A *mortgage* is a type of long-term debt secured by specific assets, obtained through private placements, suppliers or banks. Generally mortgages are repaid in *blended payments* which amortize principal and interest over the term of the loan.

Recently new types of debt have been designed to fill market gaps; one example is the *consumer savings bond* with a guaranteed minimum interest rate and guaranteed repurchase price, both of which reduce the risk of the investment compared to other alternatives available to individuals.

With minor variations, these types of debt may be found in both domestic and foreign countries, in foreign currencies or in domestic-currency debt traded in foreign markets (*Euro-Canadian debt*).

2. Borrowing costs

Debt financing incurs costs for *interest*, *transaction costs* and, if applicable, *foreign exchange*, each of which varies with market conditions and administrative and legal requirements.

2.1 Interest costs

Interest costs for a given time period are calculated as an interest rate for that period times outstanding debt principal. Interest-rate theories explain the general level of interest rates in terms of the supply and demand of loanable funds, a liquidity premium and the inflation rate; in practice, general rate levels are also affected by market expectations about economic, political and social conditions that affect dealer and investor action and consequently interest rates.

The interest rate for a debt may be thought of as having two main components: the risk-free rate for that maturity and the spread between the risk-free rate and the rate for the debt. The *risk-free rate* for a given debt term is the rate for a government debt security with that term. Risk-free rates vary with *maturity* (repayment horizon), giving a relationship referred to as the *term structure of interest rates* or, at a given time, the current *yield curve*. Interest rates most frequently increase with increasing term (referred to as an *upward-sloping yield curve*), although the yield curve may also be *flat*, *humped* or *downward-sloping*. Empirically supported hypotheses which attempt to explain the term structure include the *expectations hypothesis* (that the relationship between short- and long-term rates reflects expectations about future rate movements) and the *liquidity-preference hypothesis* (that investors need greater compensation for investing for longer terms and assuming greater risk).

The *spread* between the risk-free rate and the rate for a particular debt security reflects the credit risk of the security, in addition to depending on the debt's features as described above. *Credit risk* is a measure of the likelihood that the borrower will default on interest or principal payments when due. Relative credit risks for bonds and commercial paper are assessed by rating agencies (Standard & Poor and Moody's in the U.S. and Canada; the Canadian and Dominion bond rating services for Canada alone) and reflected in borrower *credit ratings*. Interest rates *increase* with credit risk in order to compensate lenders for the additional risk. Investment dealers use additive rate spreads whose values vary over time, tending to increase when rate levels are high and to decrease when rate levels are low.

2.2 Bond discount and premium; price volatility

As mentioned above, the price actually paid for a corporate bond, on initial sale or subsequent trading, differs from its par value according to the market interest rate for that debt at the time of the sale. The *market price* for a bond is calculated as the present value of its future cash flows (coupon payments and principal repayment) discounted at the current market rate; prices therefore rise when rates fall and fall when rates rise. The longer the maturity of a bond (holding the coupon rate constant) the greater is its price change for a given interest-rate change. Price volatility due to changes in the general rate level is called *market risk*.

Bond market prices after initial issue are of interest to borrowers primarily because (a) a corporation can sometimes purchase its own bonds on the open market for early retirement or sinking fund purposes, (b) market prices can be used to value debts at any time, as for giving a surrogate retirement cost for all outstanding debt at the end of a planning period; and (c) they reflect the evaluation given by the market to management for performance of its ongoing responsibilities.

2.3 Transaction costs

Transaction costs for issuing debt include the discount or premium on issue as well as an *underwriting commission* (fee paid to the investment dealers who market the debt) and *legal and administrative fees.* These costs vary according to the legal requirements and business conditions in the market in which the debt is issued; for example, markets may or may not require the filing of a formal *prospectus* (statement of the borrower's financial condition, debt structure and projected future operating results) before approval of a long-term public bond issue.

2.4 Foreign exchange costs

Foreign exchange costs are incurred when exchange rates change during the term of a debt issued in foreign currency. When this happens, debt transactions such as interest payments, principal payments and sinking fund payments are made in the foreign currency at a different exchange rate than was in effect when the debt was issued, resulting in a gain or loss to the borrower.

Foreign exchange rates are determined by the supply and demand of currencies, which in turn vary with international and country-specific economic and political conditions. The theory of *interest rate parity* states that if markets are efficient, foreign exchange and internal interest rates cancel out over time so that borrowing in a country with lower interest rates should result in compensating losses through exchange rate changes. This is because a higher or lower expected relative interest rate in each country is reflected in (compensating) lower or higher expected exchange rates (Viscione and Roberts 1987).

3. Uncertainty and risk in borrowing

Borrowing decisions are made in an uncertain environment, in which future interest and exchange rate movements can only be predicted with educated guesses. If these rate movements were known, long-term borrowing could be undertaken to guarantee low rates before they increase, while short-term debt could be relied on when rates are high and decreasing. In practice, however, maturity and timing decisions are made without this information.

3.1 Types of risk

For the purposes of this discussion, we define *borrowing risk* as variability in borrowing costs, either upward or downward, over the period during which debt is outstanding. (In practice, corporate borrowers are likely to be more concerned with *downside risk*, or the unfavourable (upward) variability of borrowing costs.)

Three main types of risk arise from the uncertain future market conditions described above.

Interest rate risk is the risk that borrowing costs will change along with interest rate levels. The component of interest rate risk associated with interest payments is lower the longer the average debt term and the lower the proportion of floating-rate debt used by a firm, but the market price component (primarily considered in valuing debts at the end of a planning period) shows greater risk the longer the average debt term, as noted above.

Rollover risk is the risk that funds may not be available to the corporation in the future when needed; it also decreases with longer debt terms. Rollover risk increases as the general credit rating of the borrower decreases and as borrowing is concentrated in amounts too large for markets at a given time.

Exchange rate risk arises from movements in foreign currency exchange rates and increases with the proportion of the debt portfolio in foreign currencies, assuming that all cash flows to the company are in the home currency.

Since all borrowing decisions imply acceptance of less or more risk of some or all of these types, the debt manager in borrowing explicitly or implicitly chooses the risk as well as cost behaviour of the corporation's debt. As in all reasonably efficient markets, there is a tradeoff between cost and risk so that lower-cost debt generally carries with it higher risk features.

3.2 Risk management techniques

Borrowing costs or risks can be lowered in certain circumstances by hedging or swap transactions. *Hedging* involves buying *forward* or *futures contracts* (contracts to buy or sell specified currencies at future dates at specified prices) which lock in interest or foreign exchange rates for part or all of the term of a debt; *swap transactions* are more complex arrangements in which corporate debts in two different markets are traded for lowered cost on one side and lowered risk on the other. Financial institutions can also attempt to match the *duration*, or average time before future cash flows are received, weighted by size of cash flows (Bierwag 1987), of their financial assets and liabilities so that any change in value on one side is offset by an equal change on the other (Grove 1974).

As with investing, borrowing strategies can be active or passive with regard to treatment of risk exposure. An *active strategy* will attempt to forecast future rates and to speculate on them by allowing transactions to take place at future spot rates when they are believed to be more favourable than current futures rates. A *passive strategy* will always attempt to hedge or otherwise minimize risk on the assumption that the rates available through hedging or swap transactions are the best available forecast of future rates.

To summarize, the corporate debt manager is responsible for making decisions involving selection from available debt instruments, terms and sources so as to satisfy cash requirements, minimize cost, manage risk and meet other requirements related to market conditions and corporate priorities. Actual cost is determined by future interest and exchange rate movements, market capacity and borrower credit rating; hedging or other techniques may be used to reduce interest and exchange risks by locking in known costs. The complexity of these decisions has made debt management a natural application area for modelling and decision support, both for suggesting acceptable borrowing decisions and for exploring the implications of borrowing alternatives.

4. Debt management theory and practice

If efficient financial markets are assumed, then the timing, source or other aspects of borrowing decisions should be irrelevant to a corporation's borrowing cost over the long run (Brealey *et al.* 1986). This is so because in efficient markets, individual-firm forecasts of rate movements will be no better over the long run than the market forecast, embodied in current rates; corporate borrowing cost will be determined by the underlying risk in its business operations and will reflect all market and borrower risk preferences.

Research into the theoretical aspects of corporate borrowing has concentrated on developing and analytically proving or empirically testing models explaining borrowing behaviour. Major aspects considered in this work include *capital structure* (debt/equity mix) (Modigliani and Miller 1958, 1963); *debt maturity structure* (mix among various debt terms, once the capital structure is fixed) (Brick and Ravid 1985); the effects of *agency costs* (costs due to managers acting in their own rather than the corporation's interests) (Bodie and Taggart 1978, Barnea *et al.* 1980, 1981*a,b*); and duration (Grove 1974, Morris 1976*a,b*). Because these studies rely on various limiting assumptions about market efficiencies and corporate conditions, they have little normative application to actual borrowing decisions. Portfolio analysis techniques have been theoretically applied to borrowing by Agmon *et al.* (1981) and Kalotay (1980), but difficulties in measurement of the covariances needed for their models limit their application to practical problems.

In efficient capital markets, borrowing decisions should be irrelevant to a firm's financial results as noted above. In such markets, over the long term all borrowing decisions should result in expected *net present values* (NPV's) of cash flows (cash flows discounted at the borrower's cost of debt) equal to zero. However, empirical evidence indicates (a) that corporate bond markets may be inefficient (Fabozzi and Fabozzi 1989), and (b) that firms assume either that they have superior forecasting ability or that market inefficiencies exist (Boot and Frankfurter 1972, White 1974). Borrowers therefore attempt to time their debt issues so that short-term debt is used more when rates are high and long-term when rates are low relative to long-run averages. Another commonly-accepted rule of thumb recommends that debt maturities be matched with asset lives in order to minimize risks of financing associated with particular projects (Viscione and Roberts 1987); however, Handorf (1974) found increasing use of short-term debt to finance longer term investments as rates increased during the 1970s. Some research has investigated debt covenants; for example, Thatcher (1985) linked the use of call provisions to certain agency costs. Taken together, these studies suggest that debt managers do act as though borrowing decisions are capable of improving firms' profitability and value and that they may be justified in doing so.

5 Analytical techniques for debt management decision support

5.1 Problem formulation

In the absence of a comprehensive theory of debt management, we follow the empirically-verified practice of assuming that market imperfections do indeed exist, so that positive cash flows can be generated from debt decisions and these decisions are relevant to the value of the firm. (For our analyses, we also assume that capital structure is fixed, which is realistic in the case of the Corporation since it is set by government legislation.) Because existing analytical models do not provide full normative solutions to debt problems, we incorporate known management science approaches.

Given these assumptions, the borrowing decision becomes a capital budgeting exercise (Brealey *et al.* 1986). Issuing a debt security, calling a debt issue and any other such decision is evaluated by computing the NPV of the decision's cash flows. A borrowing decision will increase the value of the firm if it leads to a positive NPV. Because in general the first cash flow is positive while the remaining cash flows (interest and principal repayments) are negative, a borrowing decision presents a reverse situation from an investment decision, in which the initial cash flows are outgoing and later returns are realized as positive cash flows.

In this analysis, interest, transaction and foreign exchange costs are reflected in the sizes of cash outflows, so that higher-cost alternatives have lower NPVs. Risk is reflected in the variation in NPVs with changing future rate assumptions.

The appropriate discount rate for debt NPV calculations is generally taken to be the firm's ***average before-tax cost of debt*** (Brealey *et al.* 1986). Because borrowing decisions affect the average debt cost, which is used to evaluate borrowing decisions, this approach is circular; evaluation using the decision's *internal rate of return* (IRR) (the discount rate at which the NPV of future cash flows equals zero) avoids the problem, but the latter has multiple solutions if cumulative discounted cash flows change sign during the evaluation period.

5.2 Analytical approaches

Borrowing takes place over time, and the total cost and risk of a corporation's debt depends on the combination of debts and hedges used over time, not solely on individual decisions or decisions at one time. (For example, the cost and risk of a single ten-year, 12% bond issue vary from the cost and risk of a series of one-year floating-rate notes.) Borrowing can therefore be viewed as a *portfolio management problem*, in which the objective is to choose a combination of debts meeting cash requirements at minimum cost and within other constraints. A range of modelling techniques has been successfully applied to debt or investment portfolio management decision support, each with its own strengths and limitations.

5.2.1 Linear and integer programming

Linear and integer programming techniques have been widely used to suggest optimal portfolios given borrower and market constraints. The following examples illustrate the range of these approaches:

1. Lee (1985) applies deterministic linear and integer programming to the *investment decision with capital rationing*, which corresponds to the borrowing problem with signs of variables and inequalities reversed. His model maximizes the net present value of a combination of potential capital projects over multiple periods, subject to cash availability constraints. The shadow prices resulting from the solution express the cost of capital rationing limitations in each period.

2. Crane (1971) uses a two-stage stochastic linear program to optimize a *bank bond portfolio* under uncertain future interest rates and cash flows. Uncertainty

is handled through scenarios in which random events with a finite number of outcomes determine values for each period. The model's decision variables are the amounts of several bond maturities bought, held and sold in each period; the model maximizes expected total profits subject to limits on downside risk (maximum loss in any period).

3. Bradley and Crane (1972, 1973, 1975, 1980) extend the Crane model to a multi-period stochastic program using a decomposition technique to reduce solution time for large problems. Their model uses any finite number of time periods, a finite number of possible yield curves realized through random events, and a finite number of security classes; buy, hold and sell decisions are made to maximize the expected future value of the portfolio at the planning horizon. Constraints in addition to the usual nonegativity conditions specify inventory balancing (linking amounts bought, held and sold from period to period); cash flows available for investment; and limits on net capital losses which express management's aversion to downside risk. Their decomposition technique enabled solution of models with 190 constraints in 213 seconds on an IBM 360/65.

4. Crane *et al.* (1977) used a similar but deterministic model to manage ***bank borrowing*** over eight quarters by selecting from various maturities to meet stated cash requirements. This model minimizes the net present value of average interest costs plus terminal debt market value; the latter term is included as a proxy for debt costs occurring after the end of the planning period. Decision variables are amounts of each maturity sold in each period; constraints include limits on amounts of specific maturities and limits on amounts maturing in specific periods.

5. Lane and Hutchinson (1980) describe a stochastic programming model for ***determining the pattern of purchases and sales of securities*** which maximizes the expected return for a bank portfolio of certificates of deposit. It is similar in approach and structure to the models outlined above; in addition, it incorporates

recourse through penalty borrowing. Future rates are specified as branching trees of significant events which correspond to financial managers' scenario descriptions.

6. Booth and Koveos (1986) use a two-stage stochastic programming model to *suggest hedging strategies for bank asset-liability management.* It selects from short- term securities, long-term securities and futures contracts. Its objective is to maximize the expected ending balance sheet value of shareholders' equity, equivalent in the model to maximizing expected profit.

7. Shapiro (1988) applies stochastic programming with recourse to manage *dedicated bond portfolios*, in which cash flows from assets are structured to match forecast future liabilities. This model minimizes the present cost required to meet future cash requirements; a number of constraints such as limits on issue sizes may be used to describe market conditions or portfolio managers' operating policies.

8. Mulvey and Vladimirov (1988; 1989*a*,*b*) express multiperiod financial portfolio management (asset allocation) problems as dynamic stochastic network models which maximize a utility function subject to inventory, cash availability and other constraints.

9. Deterministic linear programming based on scenarios is becoming popular as a tool for commercial firms advising clients on *investment portfolio management.* Adamidou *et al.* (1989) and Dembo (1989) describe portfolio optimization approaches in which a deterministic problem is solved over a specified planning horizon for each of a number of future interest rate scenarios; a subsequent (proprietary) optimization technique is used to produce a portfolio that meets specified risk-return tradoffs such as minimizing downside deviations from target returns on all scenarios.

5.2.2 Dynamic programming

Dynamic programming has also been applied to debt management; in particular, it has been used to model *bond refunding decisions* by Elton and Gruber (1971), Kalymon (1971) and Boyce and Kalotay (1979). The last model, for example, chooses the *optimal timing of a call decision* for a long-term bond issue based on stochastic future interest rates.

5.2.3 Simulation and projection

Simulation and projection are widely recognized as techniques for evaluating complex investment portfolios in more detail than is generally included in optimization models, as illustrated in the following examples.

1. Bradley and Crane (1975) describe the use of simulation in initial attempts to *manage bank bond portfolios*. Their model tested specified portfolio strategies, expressed as maturity mixes such as 'equal proportions of short-term and long-term bonds'. A number of future economic scenarios are then generated; for each simulation pass, one such scenario is randomly selected, an aggregate portfolio of maturity categories is created and its results (final market value, interest income and unrealized gains and losses) computed. Distributions for these indicators thus describe both the performance of the strategy and variation in performance under varying future conditions.

2. Frank and Schnabel (1983) project the net present cost of borrowing alternatives based on future interest rate projections which simulate past rate movements. The system also produces the interest rate increase equating two alternatives. This system is used to assist a financial manager in *choosing the timing of new borrowing*.

3. Howard (1986) uses stochastic simulation of cash flows to *evaluate individual borrowing and hedging decisions and strategies*. Interest rates are generated

through a random walk procedure. Cash flows are measured using net present values or internal rates of return; suggested *risk measures* include NPV or IRR standard deviations and the probability that these measure rise above specified values. This paper also illustrates presentation of results for managers in a variety of graphic and tabular formats.

4. Several sources describe the use of Monte Carlo simulation for *valuations of individual securities or options* for which cash flows vary with interest rates. Examples of these are *mortgage-backed securities* in the U.S. (Zenios 1989), which are subject to repayment risk if interest rates rise; *single-premium deferred annuities*, which contain certain options sensitive to interest rates (Asay *et al.* 1989); and any securities with options such as calls (Prisco 1989). For these applications, large samples of future rates are generated, cash flows generated and theoretical option-adjusted spreads calculated. Differences between these spreads and actual market prices indicate profitable investment opportunities.

As noted earlier in this chapter, simulation and projection techniques allow detailed calculation of the expected consequences and risks of alternative decisions, but they do not provide decision criteria. Instead, alternatives must be eliminated or selected using management's subjective criteria or analytical techniques such as Markowitz portfolio analysis (Markowitz 1959), which develops an *efficient frontier* of portfolios with equivalent risk/return characteristics.

5.2.4 Multiple-model combinations

The combined use of optimization, simulation and projection models is not found in the portfolio management modelling literature, although it is clear that they offer complementary advantages. In another problem domain, Hamilton and Moses (1973, 1974) demonstrated this approach in a decision support system for high-level *corporate planning* in which a deterministic integer program with approximately 1,000 decision variables and 750 constraints was supplemented by a detailed cash flow projection model as well as econometric forecasting capabilities. Their optimization model maximized a linear approximation of earnings per share and ran in 15–30 CPU minutes on a Univac 1108 system. The projection and simulation models generated detailed profit projections and probability distributions which were used to support constraint setting in the optimization.

6. Debt management at the Corporation

The Corporation is a Canadian provincially owned electric utility. Its 1990 annual report (Nova Scotia Power Corporation 1990), reports its total assets as at March 31, 1990 as $1.716 billion and total debt net of sinking funds as $1.673 billion. (As a Crown Corporation, it is 100% debt financed; the difference between assets and net liabilities is accounted for by sinking funds and small equity balances.) Its only revenue source, apart from miscellaneous short-term investments and other minor activities, is payment for electric power service from individual and corporate customers; power rates are regulated by a provincial Public Utilities Board. It is *exempt* from all income taxes.

Borrowing is a major concern for the Corporation since it is the sole source of financing for capital projects and operating deficits. Interest costs in fiscal 1990 were $152 million — 26.1% of total expenses and the second highest expense after fuel costs. Figure 3.1 gives a profile of the Corporation's long-term debt as at March 31, 1990.

Interest cost to the Corporation is determined by (a) risk-free market rates as reflected in Government of Canada securities, and (b) the perceived risk of Corporation debt, measured by the Corporation's credit ratings and reflected in the spreads between Corporation and government debt. Because all the Corporation's debt is guaranteed by the Province of Nova Scotia, its credit rating is the same as that of the Province (A– at this writing). Should market concern for the Province's economic performance and heavy debt load lead to a reduced rating, interest cost for all *new* debt would immediately increase.

Long-Term Debt
thousands of dollars

Long-term debt which is guaranteed by the Province of Nova Scotia is expressed in Canadian dollars. The rates in effect at balance sheet date summarized by years of maturity and by currency payable are as follows:

	1990				1989	
Years of Maturity	Principal Outstanding			Weighted Average Coupon Rate	Principal Outstanding	Weighted Average Coupon Rate
	Canadian	Foreign	Total	%	Total	%
1990					$ 245.108	
1991	$ 20.090	$ –	$ 20,090		20.143	
1992	178.355	62.536	240,891		231.139	
1993	129.826	–	129,826		130.222	
1994	15.759	–	15,759		53.836	
1995	51.190	4.447	55.637		–	
1 5 Years	395.220	66.983	462,203	10.68	680.448	10.26
6 10 Years	127.148	81.914	209,062	9.23	165.364	8.60
11 15 Years	559.365	58.510	617,875	11.02	571.443	10.83
16 20 Years	54.930	87.765	142,695	10.21	244.366	10.99
21 25 Years	350.971	–	350,971	11.08	151.938	10.83
26 30 Years	150.000	–	150.000	10.25	–	–
	$1.637.634	$ 295.172	$ 1,932,806	10.64	$1.813.559	10.43
Less sinking funds	257.311	143.084	400.395	–	353.784	–
	$1.380.323	152.088	1.532.411	10.64	$1.459.775	10.43
Currency payable:						
Canadian dollars			$ 1,380,323	11.06	$1.257.507	10.91
United States dollars			89,552	9.22	144.820	9.04
Swiss francs			62.536	4.88	57.448	4.88
			$1.532.411	10.64	$1.459.775	10.43

Figure 3.1. Nova Scotia Power Corporation Debt as at March 31, 1990.
Source: Nova Scotia Power Corporation (1990).

The Corporation's Treasurer is responsible for management of the Corporation's debt. *Cash requirements* are determined outside the Treasury by management and regulatory board decisions about power rates, operating expenses and capital projects. The Treasurer must determine the *mix of debt* used to satisfy these financing requirements at minimum long-term interest cost net of investment income from sinking funds and temporary cash surpluses (Nova Scotia Power Corporation 1985). He also tries to maintain *stable cash flows* by distributing debt maturities and interest payments over time wherever possible.

The Treasurer and his staff manage the Corporation's debt with the help of a financial advisory firm which also acts as lead *underwriting manager* for long-term bond issues. This company provides economic data and market commentaries in a variety of newsletters, general and company-specific rate forecasts, and advice on specific issues as requested by the Corporation.

6.1 Debt sources

The majority (74.5% in 1988–89) of the corporation's debt is in the form of *public bond issues* in the Canadian and US markets. These bonds finance the Corporation's capital projects; they typically have sinking fund and call features and are issued in terms of from 15 to 25 years. In 1988–89, sinking fund balances equalled $353.8 million or 19.5% of the Corporation's long-term debt outstanding. Other long-term financing (7.8% in 1988-89) is obtained through long-term goverment notes. Long-term financing for equipment (.07% of long-term debt in 1988-89) is often obtained through *supplier notes* or *mortgages*, arranged as part of the equipment purchase.

In 1986 through 1989, the Corporation issued series of *savings bonds*. These were intended to tap the small investor market and sold in multiples of $100 for a 5-year term, transferrable at any time and redeemable at par at each anniversary of issue; they require an ongoing sinking fund to finance their retirement. Interest at a rate slightly higher than the 1 year GIC rate is payable annually. In 1988–89, 17.6% of the Corporation's long-term debt or $321 million was in the form of savings bonds.

There was no savings bond issue in 1990.

Short-term funds other than trade accounts payable are obtained from a *line of credit* arranged with five Chartered banks, at the prime rate or slightly less, and from unlimited credit with the Province of Nova Scotia, at about 1% below prime. Use of the bank line is minimized due to its higher cost, and a ceiling amount of $50–100 million is maintained, after which long-term debt is issued.

6.2 The borrowing decision process

The decision process within the Corporation prior to a bond issue emphasizes monitoring rate movements and spreads and attempting to time debt issues to coincide with the availability of favourable rates. Little formal long-term debt planning is done, although the Treasury department has developed heuristic guidelines for operational borrowing decisions such as generally attempting to lengthen the average term of their debt. Within these guidelines, the Treasurer decides on the timing, amount, currency, market and features of each new bond issue and each possible refinancing as it is needed. Limited use of forward contracts to hedge US dollar interest payments was begun in 1990.

Before bonds are issued, the Corporation's financial advisors are consulted for assessments of market conditions, future interest and exchange rates and recommendations as to the 'best' financing alternative for present conditions. These assessments identify short-lived opportunities for maturities and currencies in which spreads are less than expected. Costs for alternatives under consideration are compiled and summarized by staff for the Treasurer using spreadsheets; alternatives may be compared using cash flows discounted at rate used for evaluating capital projects, which is an estimate of the current average cost of financing. A recommendation is decided on by the Treasurer, based on these supporting analyses, his knowledge of current market conditions and his experience. The final decision is made through a series of discussions and approvals by the Chief Financial Officer, President and Chairman in consultation with the financial advisors and Provincial representatives. Long-term borrowing and refinancing decisions must be formally approved by the Board of Directors and the Province. The entire process for a new long-term issue could in theory be completed in just a day, although it is longer in foreign markets with additional administrative requirements. In practice, the decision and approval process generally takes several weeks, making it difficult for the Corporation to take advantage of windows of opportunity for favourable borrowing.

Once a bond issue is approved, the marketing and sale of the issue is carried out by four underwriting managers and an 18-member banking group with little day-to-day involvement by the Corporation. Subsequent administration activities including ***debtholder registration*** (recording debt owners to receive future payments) and payment of interest are handled directly by the Corporation.

For a savings bond issue, the key decisions to be made are (a) its interest rate and (b) the amount to be offered in a given year, both of which are set by weighing anticipated savings bond costs against those of other alternatives and considering other factors such as the minimum amount to be issued to maintain subsequent tradability for investors in the savings bond market.

The main activity associated with short-term borrowing is monitoring cash flows to minimize use of short-term credit; little analysis is required.

7. The role of a decision support system

As can be seen from the above description, borrowing decisions at the Corporation are analyzed on an 'as needed' basis with heavy reliance on experience, heuristics, outside advice and informed estimates of future interest rates. Formal analysis consists mainly of cost projections for individual decision alternatives; no attempt is made to formulate long-term plans or borrowing strategies, and no effort is made either to analyze borrowing as a portfolio problem or to incorporate uncertainty in future rates into the analysis that is done.

The frequency of new long-term debt issues depends largely on the Corporation's capital projects but has varied from one to four new savings or corporate bond issues per year during the late 1980s. (In periods of falling interest rates such as the mid-1980s, one or two call and refinancing transactions are also considered each year.) These transactions typically vary in size from \$50 to \$150 million, giving a high potential benefit even though the number of transactions is small, if interest costs and risks are reduced by improved decision support.

As outlined in a previous section of this chapter, models have been developed which could handle the portfolio and contingency aspects of the borrowing problem. Two types which are particularly suitable for application at the Corporation are (a) stochastic linear programming models for determinimg the optimal term/market/features mix and (b) simulation and projection models for predicting the cost/risk behaviour of individual debts or groups of debts. Use of these model types in an integrated manner within a domain-specific decision support system would improve the borrowing decision process for the Treasurer in three major ways:

1. It would facilitate the development, analysis and refinement of *portfolio-oriented contingent borrowing plans* over a *rolling planning horizon* such as the five years used for detailed budgeting within the Corporation.

2. It would incorporate *uncertainty* into the analysis of both borrowing plans and individual borrowing alternatives, so that the range of possible results borrowing decisions could be readily appreciated.

3. The use of cash flow projection modelling with deterministic future rate assumptions would provide the *detailed analysis of individual borrowing alternatives* that is carried on now, without the need for developing individual spreadsheets for each decision.

These models cannot, however, be easily implemented within the Corporation Treasury since Treasury staff are not trained or experienced in their use or interpretation. Extensive assistance (human or machine-based) is needed for them to be used productively. Providing (a) a comprehensive modelling facility for debt planning and (b) at least some of the assistance needed for its use by financial managers are the overall goals of this research.

4
System Overview

The system designed and prototyped for this research project is called MIDAS, an acronym for *Manager's Intelligent Debt Advisory System*. MIDAS is designed to provide a variety of modelling functions in support of borrowing decisions made by the Corporation Treasurer. It expands on his current modelling facilities in three ways: first, by modelling full-scale long-term contingent borrowing plans as well as single debt decisions; second, by including explicit consideration of risk as well as borrowing cost; and third, by providing intermediary assistance allowing him or his staff to directly set up, use and interpret the results of analytical processes. Overall, it is designed to improve his decision making by allowing him to generate and test more and better decision alternatives and to develop greater insight into the implications of possible choices both from a short-term and a long-term perspective.

While these goals are not unusual for a DSS, MIDAS is unique in that it delivers these capabilities using an integrated collection of object-oriented, rule-based and standard FORTRAN modules which allows much greater flexibility, economy of representation and 'intelligence' than are incorporated into most existing financial DSSs. This chapter first describes the conceptual debt planning approach supported by the system, continues with the major functional goals guiding system design and concludes with a description of the system's architecture, the roles of its key components and the hardware/software configuration used in its implementation.

1. The decision process: model-based debt portfolio planning

As noted in Chapter 3, a portfolio management approach to debt planning captures the interactions and dependencies among debts which determine total debt cost and risk over a given time period. Using this approach, the debt planner builds a (hypothetical) future debt contingency plan and associated portfolio which meet corporate objectives. This type of problem is referred to as *configuration* in expert systems terms and as (contingent) *allocation* in optimization; in either case, a combination of debts is specified to satisfy desired constraints in all contingencies, the main constraint being meeting cash requirements over future time periods. The problem can be solved heuristically, as illustrated for other domains by XCON (Waterman 1986) or PLANET (Dhar and Pople 1987); alternatively, it can be done algorithmically using the optimization approaches described in Chapter 3.

For this project, the decision was made a priori to incorporate proven modelling (optimization, simulation and projection) techniques where appropriate, using heuristics to enhance rather than replace modelling. The models and heuristics are used to carry out steps in a hierarchical planning process which, under user control, progresses in stages from simplified to more realistic borrowing plans (Figure 4.1). Following the initial description of a planning problem, a contingency plan incorporating major decisions is developed using a dynamic stochastic programming model. Second, the resulting optimal but incomplete plan is refined heuristically by adding details not considered in the first stage, which may make the plan no longer optimal but which must be included in a complete contingency plan. This stage may result in several alternative plans which must be tested further. Third, simulation and projection models are used to compute the future cost and risk behaviour of alternative contingency plans; the plans are then evaluated according to corporate objectives and a set of preferred plans selected for further consideration and possible implementation.

Broadly, this process uses a generate-and-test approach in which the optimization model is used to prune the initial search space to one plan before alternatives are generated heuristically, tested and evaluated. Although analysis progresses sequentially, each step results in feedback which may result in modification of inputs and reexecution of a modelling procedure; the order of process execution, while suggested, is always under the control of the Treasurer. This process is a significant extension of the short-term, single-decision, heuristically oriented one now used in the Corporation.

Figure 4.2 presents a logical dataflow diagram for this model-based planning process. This diagram identifies individual data sources, sources of expertise, processing steps and data and knowledge movements without regard for whether they are implemented within or outside a computerized system. The process' goal is to develop a borrowing plan which meets problem requirements in all contingencies, where a *borrowing plan* is defined as a set of contingent borrowing actions over the planning period; each *borrowing action* specifies the type, source, term, borrowing date, features, amount and, if applicable, call date for a debt to be initiated during the planning period in a specific contingency. The full borrowing contingency plan must obtain sufficient cash through borrowing to meet stated cash requirements during the planning period in every set of future conditions; it must also satisfy other contingent requirements reflecting current market conditions, future rate assumptions, debt availability, cost constraints, risk constraints and other conditions specified by the borrower.

This planning process involves the following steps:

Process HDP1. Describe the current problem

The first stage in the process is to precisely describe the current problem under investigation. From a domain viewpoint, this consists of accurately and completely determining and describing the borrower requirements, market conditions, future rate assumptions, existing debts and other domain conditions that guide and constrain the current plan. From a modelling and reasoning viewpoint, these translate into the assumptions, constraints and initial conditions to be incorporated into models and rule-based reasoning.

In this step the types of debt are enumerated from which the future contingent

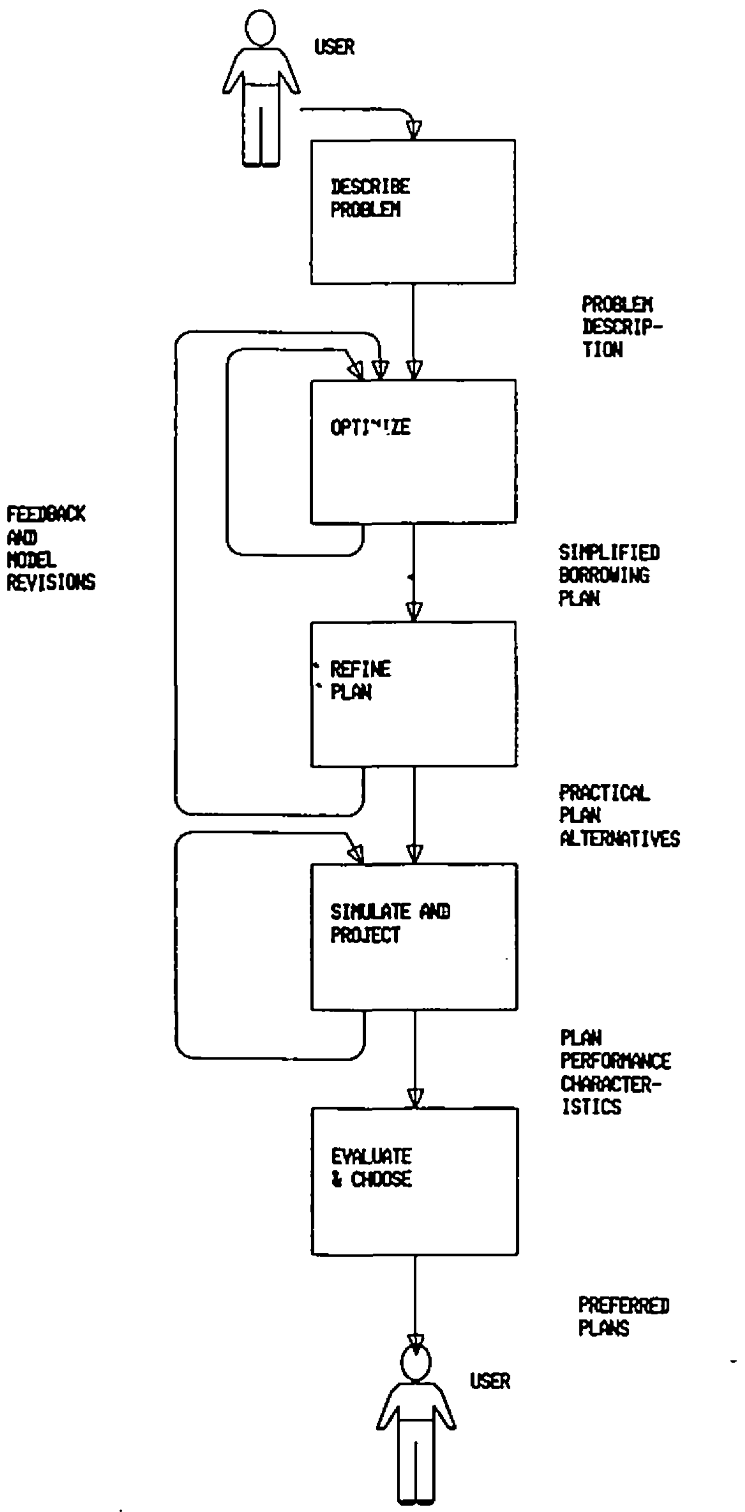


Figure 4.1. The Model-based Debt Planning Process

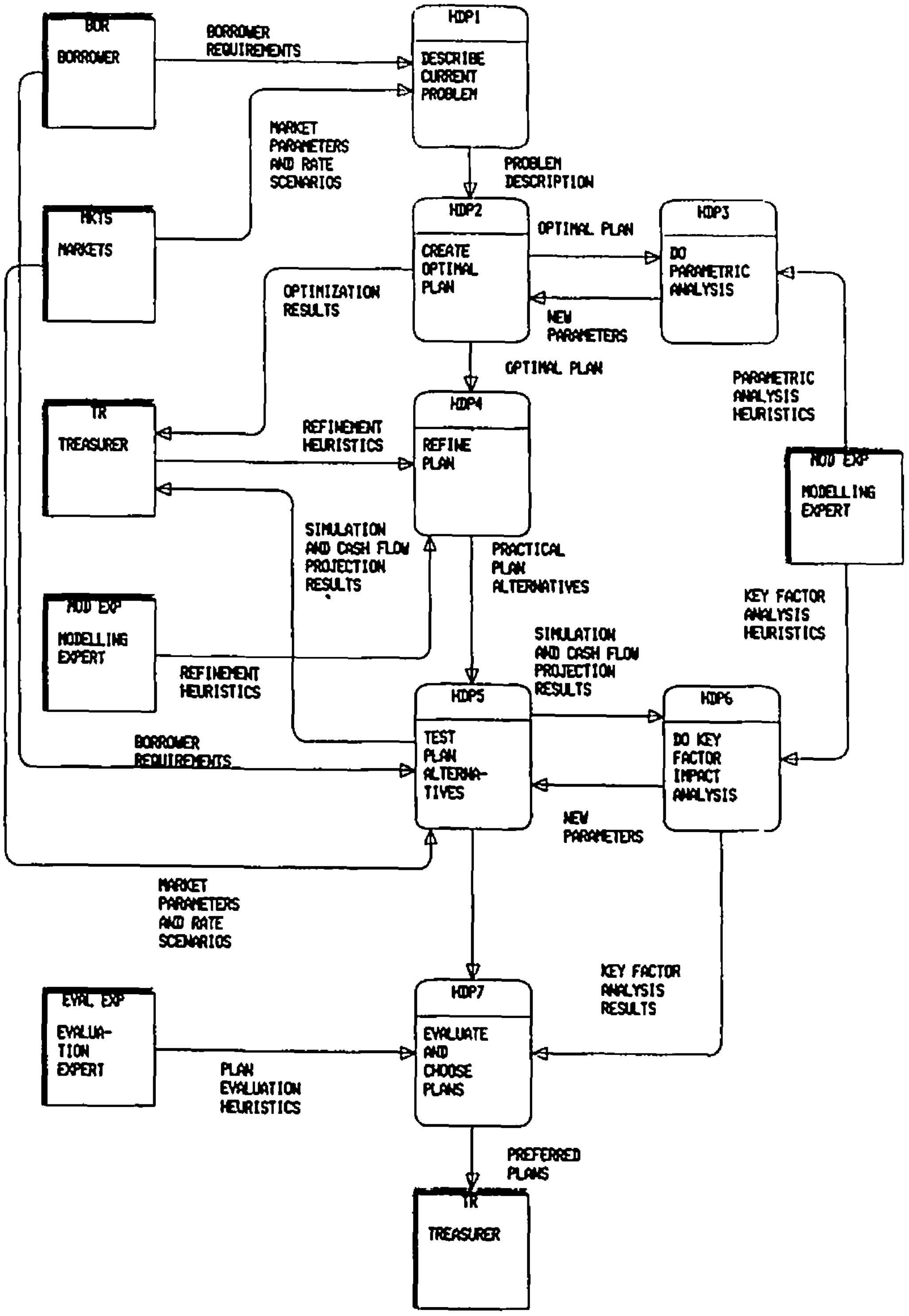


Figure 4.2. Hierarchical Debt Planning: Dataflow Diagram

portfolio can be built. The set of debt types available in all time periods forms a choice set of borrowing alternatives from which the contingent debt portfolio will eventually be constructed.

Debt types are specified by the Treasurer within the limitations of current market and modelling conditions. Each debt type translates into a set of decision variables for optimization modelling purposes.

Process HDP2: Create an optimal plan

Using the current borrowing alternatives, assumptions and constraints, a contingency plan is generated which meets defined requirements while minimizing the expected ending value of the debt portfolio. (The rationale for this objective function is discussed in Chapter 5). Because of size and complexity limitations in the optimization process, the resulting contingency plan will not be detailed or realistic enough to be implemented without modification; however, it provides a starting point from which to construct more practical plans.

Process HDP3: Do parametric analysis on the optimal plan

Following the optimization process, parametric analysis is carried out on the initial contingency plan to analyze the plan's behaviour as key assumptions vary. Parametric analysis results are summarized and interpreted qualitatively to give some insight into plan strengths and weaknesses.

Process HDP4: Refine the plan

Using the initial plan specification, parametric analysis results and plan refinement rules, the initial contingency plan is modified to be more realistic and robust. The refinement process generates multiple plan alternatives which may no longer satisfy problem requirements (for example, a plan may not generate sufficient cash in some time periods) and for which performance results are not yet known.

Process HDP5: Test plan alternatives

Future cash flows, costs, risks and behaviour with respect to constraints are simulated stochastically and projected deterministically for each plan alternative under current market conditions and rate assumptions.

Process HDP6: Do key factor impact analysis on simulation results.

In this step, critical factors and time periods determining simulation and projection results for each contingency in each plan alternative are identified; plan results are then tested to assess their sensitivity to variations in these factors.

Process HDP7: Evaluate plan alternatives and choose the preferred plan or plans.

Based on simulation results, sensitivities and evaluation criteria as defined in the problem description and by the Treasurer during system use, alternative contingency plans are rated and compared. The preferred plan or plans are chosen for further consideration.

2. Decision support system functional requirements

Complete functional requirements for MIDAS include comprehensive support for both the spreadsheet analysis now used in the Corporation and the model-based planning process outlined above. They extend beyond the scope of this project and are intended to be developed over several years on an evolutionary basis. The initial prototype design and implementation cover key functions which were identified relative to immediate Corporation needs, research priorities, availability of needed expertise and the estimated difficulty (translating into time and cost) of design and implementation.

The system is designed to support decisions by a user who is a financial expert but not an expert in the construction, manipulation or interpretation of models other than simple spreadsheets. It therefore hides these model management details from the user while allowing him great flexibility in defining and modifying problems to be analyzed.

Major system requirements are listed in Figures 4.3 and 4.4 along two dimensions: functions and problem scope. An initial set of requirements is given, with notations as to which are included in the initial design and in the initial prototype system.

At its highest level, MIDAS is designed to assist in developing acceptable contingent borrowing plans as described above. At lower levels, partial portfolios or individual debts can be simulated and compared with randomly generated or deterministic future rates. This provides support for individual decisions similar in functionality to the individual-spreadsheet approach already in place but with the addition of random-rate simulation.

3. Design principles

MIDAS is designed using four principles derived from known business system and expert system development principles, functional requirements, domain characteristics and experience gained during the design and prototyping process. (See Chapter 9 for a description of the evolution of the system during prototyping.) These principles are:

1. *Frame-based knowledge representation.* Because of the diversity of functions it is designed to support, 'knowledge' in MIDAS refers to a diverse collection of facts, rules and modelling procedures. MIDAS represents these using various representation and reasoning/control techniques but relies primarily on frames and object-oriented programming. MIDAS objects are defined in inheritance hierarchies in which generic object classes are defined, specialized and instantiated as classes, subclasses and instances. Objects contain attributes describing their properties and relationships; attributes and their values are inherited downward through the hierarchy unless overridden. The frames may also contain meth-

Notation:

* included in design outlined in this dissertation
** included in system prototype

1. Problem specification

** – form-based input for all problem description data
– comprehensive input error and consistency checking
– automated interface to Corporation debt database to maintain a current description within the system of the existing debt portfolio

2. Borrowing alternative generation

* – automated borrowing alternative generation based on minimal form-based user input
– heuristic consistency checks against market availability conditions

3. Optimization

* – model formulation and instantiation: automated generation of optimization input matrix from problem description
** – automated solver call and matrix submission
** – solver solution of stochastic linear program
** – automated retrieval of solver output
* – integration of solver output into the system's problem state description
* – integration of optimization results into plan refinement and simulation/projection components

4. Parametric analysis

* – heuristic identification of key result factors
* – heuristic generation of suggestions for parametric analysis
* – automated parametric analysis on user request
* – automated summary and storage of parametric analysis results

5. Heuristic plan refinement

** – heuristic generation of alternative plan refinements from optimization model output
** – integration of plan refinement results into the optimization and simulation/projection modelling components

6. Simulation and cash flow projection

** – for deterministic cash flow projection: calculation of cash flows, NPVs, IRR (if appropriate) and other performance indicators for individual financial instruments and portfolios given mean rate forecasts
** – for stochastic simulation of cash flows: output of distributions for NPVs, IRR, and other performance indicators from randomized future rates based on mean rate forecasts
– for stochastic simulation: distributions for operating cash flows period-by-period

** – model formulation and instantiation: automated generation of simulation and projection models from problem and plan descriptions
** – automated model execution on user request
** – automated summary and representation of model results
* – seamless integration of results into sensitivity analysis and evaluation components

7. Key factor impact analysis

* – heuristic identification of key result factors
* – heuristic generation of suggestions for key factor analyses
* – automated key factor analysis on user request
* – automated summary and storage of key factor analysis results

8. Plan evaluation and choice

– application of selected multiple-criteria decision models to simulation results
– heuristic plan evaluation and choice suggestion

9. Result explanation and interpretation

* – Heuristic identification of key factors determining optimization results
* – Heuristic identification of key factors determining simulation and projection results
– Integration of the ANALYZE program for qualitative explanation of optimization results
* – Qualitative explanation and interpretation of overall simulation and projection results for individual borrowing plans
– Qualitative explanation and interpretation of differences in simulation and projection results for pairs of borrowing plans
– Qualitative explanation and interpretation of parametric analysis results
– Qualitative explanation and interpretation of sensitivity analysis results
– Model calculation trace on user request

10. Overall system control and support

** – pop-up menu task selection
** – heuristic user guidance on task execution order
** – tabular output
** – graphic output
* – text output for explanations
– context-sensitive help facilities

Figure 4.3. MIDAS Functional Requirements

Notation:

* included in design outlined in this dissertation
** included in system prototype

1. Planning period:

** – up to thirty years
** – quarterly periods
** – yearly periods
– a combination of both quarters and years, e.g. a planning period of four quarters and four years

2. Markets

** – Canadian
** – US
– Euro-Canadian
– UK
– Swiss
– German
– Japanese

3. Rate scenarios

** – single paths
* – branching probability trees

4. Financial instrument types

** – Corporate bonds, no special features or covenants
** – callable bonds
** – bonds with sinking fund provisions
– redeemable bonds
– extendable bonds
– consumer savings bonds
– long-term notes
– mortgages
** – bank credit lines
** – provincial credit lines
– new debt types created from components
– short-term and long-term investments

7. Hedging capabilities

– interest rate hedges
– exchange rate hedges
– swap contracts

Figure 4.4. MIDAS Problem Scope

ods (LISP procedures) specifying object operations; like attributes, these are inherited unless modified or overridden. This representation allows a modular, non-redundant specification of entities, relationships and behaviours which is especially useful for portfolio management problems since portfolios are, in fact, combinations of diverse objects.

2. *Object-oriented modelling.* Object-oriented programming allows models to be defined as collections of submodels which carry out operations specified by object class and which interact by sending messages to LISP methods in other objects. Because methods can be defined with the same name but unique behaviours for each object class, extremely flexible dynamic portfolio models can be developed as varying collections of specific-purpose objects, linked together by generalized control methods.

3. *Spreadsheet-oriented financial model structure.* Individual coefficient, cash flow and performance indicator calculations are organized conceptually as 'cells' in a spreadsheet, where each cell handles a single calculation for a single time period. In MIDAS' object-oriented programming context, each cell calculation is a single method. This approach adds modularity to the method structure, simplifying maintenance of the large numbers of LISP methods involved in the system.

4. *Separation of knowledge and control/reasoning.* This is widely regarded as one of the defining characteristics of expert systems and is the means by which complex rule-based knowledge is able to be modularized and maintained. Here the principle is extended to include generalized factual and conceptual knowledge represented in frames and manipulated either by rules or by procedural code; advantages of this approach include ease of documentation, understanding, development and system maintenance.

5. *Integration through an underlying domain representation.* Domain knowledge is required in order for a specific DSS to represent problem descriptions and map them into appropriate model instances, as noted in Chapter 2. MIDAS' design

uses a common domain representation based on the user's problem description to organize both domain and modelling knowledge and to represent all model and rule results in a common set of domain objects. This approach simplifies system development by providing a representation common to both the user and the developer, and it provides a dynamic natural correspondence between problems and model instantiations which allows straaightforward reconfiguration of multiple models.

MIDAS is described here as a system which uses two distinct model types to analyze common debt portfolio problems. An alternative view is that of a single underlying model (set of mathematical and domain relationships) which is analyzed using multiple solution techniques (optimization and simulation). This view follows that of structured modelling, which attempts to build abstract model descriptions solved by various operators; conceptual and technical difficulties must be solved before it is successfully implemented on model types other than optimization (Dolk 1990; Dolk and Kottemann 1990).

4. System architecture

4.1 Functional view

Viewed from a functional perspective as in Figure 4.5, the system's proposed architecture reflects the hierarchical model-based decision process previously outlined, incorporating modelling or heuristic subsystems for each hierarchical planning step. All modules operate under the control of a menu-driven interface and access a common system knowledge base of conceptual knowledge and control rules. An *initialization module* requests corporate goals, operating parameters and assumptions including future rate scenarios, guiding the user in entering input and commenting on input consistency and reasonableness based on modelling and economic knowledge in the knowledge base. The *stochastic programming model* is then instantiated with variables, constraints and coefficients calculated from the current parameters and

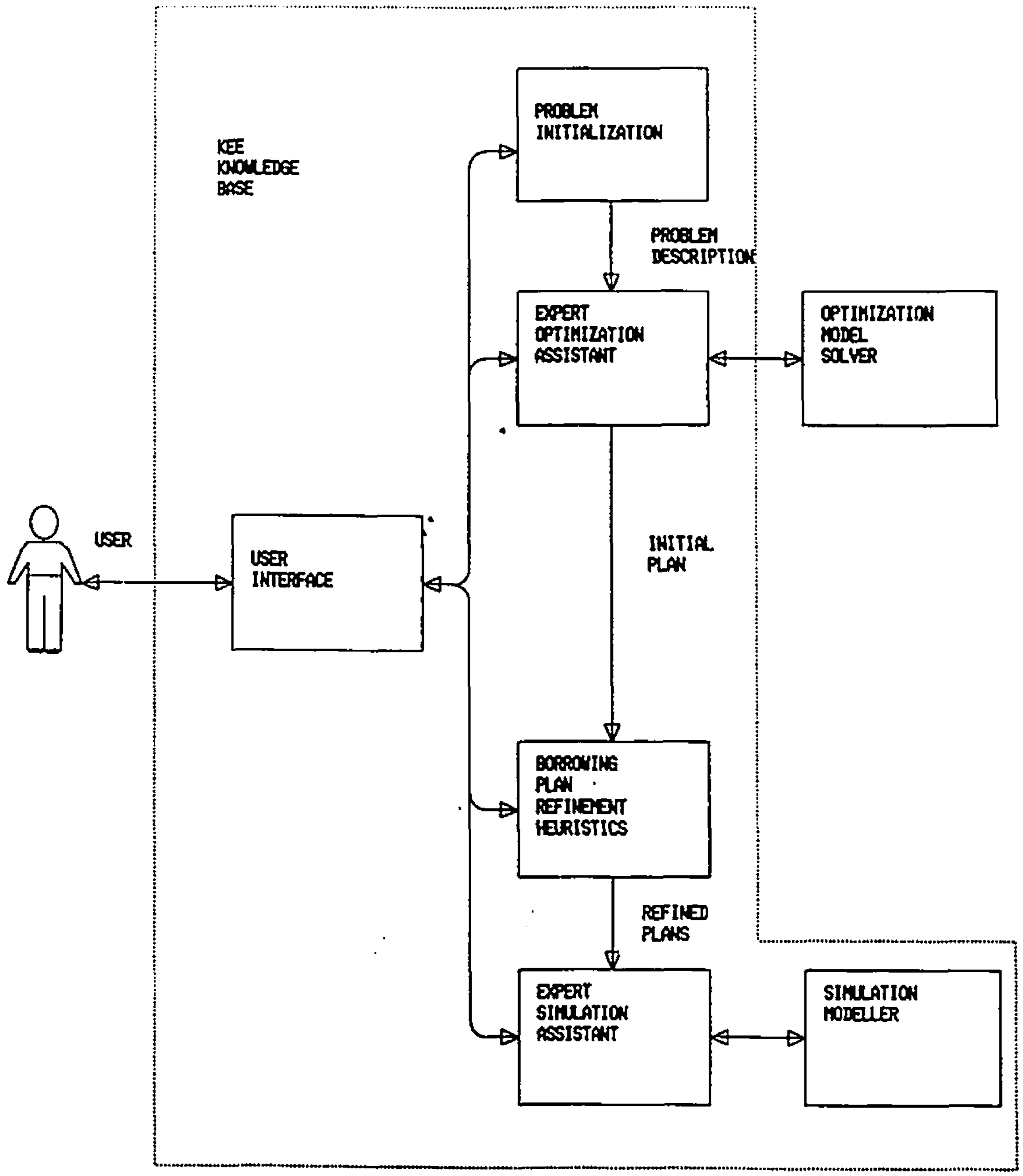


Figure 4.5. System Architecture: Functional View

solved to produce an initial borrowing plan. A *stochastic modelling assistant* formulates the model, reviews model output, responds to user requests for explanations, comments on results, identifies and tests appropriate sensitivities, recommends and makes model modifications and reruns the model on user approval. When stochastic modelling is completed, rules expressing *plan refinement heuristics* are used to construct alternate borrowing contingency plans. These are then *simulated stochastically* using rates generated randomly from distributions consistent with the scenarios used initially and *projected deterministically* using the mean-rate scenarios. A *simulation assistant* then repeats the explanation/analysis/suggestion/refinement loop with the simulation and projection model results. The resulting alternative plans are *evaluated* with respect to corporate goals and certain plans are retained for consideration by the debt manager.

With the exception of the optimization solver, all functional subsystems are designed for integrated object-oriented and rule-based implementation within a common *knowledge base* (memory-resident object, method and rule representation) on a single hardware platform. For solution efficiency, the solver may be a distributed component on separate hardware, in direct communication and under the control of the knowledge base (see further discussion later in this chapter).

4.2 Object-oriented view

Figure 4.6 gives a diagram of the system viewed as a collection of cooperating objects. (Section 2 of Appendix C describes the object diagram conventions used in this dissertation.) Within this diagram and the more detailed object diagrams in subsequent chapters, rounded rectangles represent object groups, squares represent system components outside the knowledge base, solid arrows represent data flows and dashed arrows represent messages between object groups.

Viewed in this manner, MIDAS consists of three subsystems grouped by their knowledge types and functional roles within the system. Its *modelling subsystem* represents both the problem domain knowledge and modelling expertise necessary

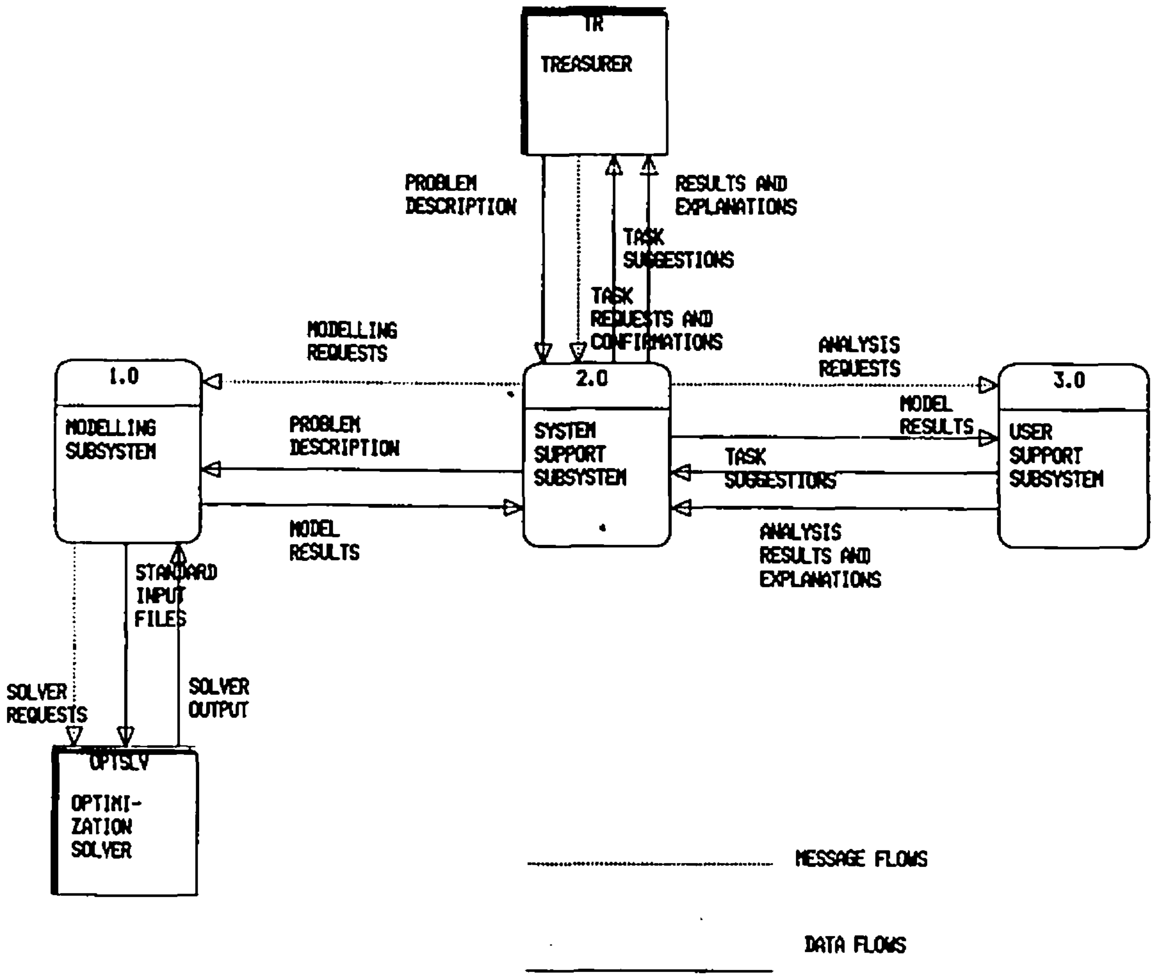


Figure 4.6. System Architecture: Object-Oriented View

to describe entities in the problem domain, model the problem domain using optimization, simulation or projection, and store the common problem states referred to in all system operations. Its *user support subsystem* stores model management knowledge and performs the intermediary functions which guide the novice user in operating, manipulating and interpreting the system's models; its objects comprise the optimization and simulation assistants previously referred to. Finally, its *system support subsystem* organizes the knowledge and procedures necessary for generalized system control and support functions such as windowing, menu generation, output management and presentation, system housekeeping and management of task order and execution.

The three groups of objects generally work in a relationship in which users interact with system support objects, which in turn request modelling in optimization, simulation or projection mode from the modelling subsystem. On completion of a modelling operation, results are presented by system support objects; user support objects monitor results and suggest further actions to the user, who may accept, reject or modify the suggestions and carry out further analysis. Overall control is handled by a top-level system control rule set and frame representations of suggested task sequences. Detailed descriptions of the structures and functions of the three subsystems are given in Chapters 6 through 8 of this dissertation, following detailed specification of the system's models in Chapter 5.

5. Hardware/software configurations

MIDAS software is implemented primarily in the KEE expert systems environment, which provides integrated frame-based knowledge representation, object-oriented programming and rule-based reasoning (Intellicorp 1988*a*). Financial calculations for coefficient generation and simulation are written in LISP, stored as methods in frames and executed within KEE. The stochastic programming model standard input generator is also a collection of LISP methods; the model solver is the FORTRAN program MSLiP (Gassmann 1989*b*) which runs on either the MicroExplorer or the MicroVAX II at the request of MIDAS methods within KEE objects. Communications between the Explorer or MicroExplorer and the MicroVAX II are handled by DECNet communications software on both systems.

The prototype system is implemented on the following hardware configurations:

1. A Texas Instruments Explorer with 4 megabytes (mb) of RAM and 200mb of hard disk storage capacity, running the Explorer operating system, DECNet, Common LISP, KEE and the MIDAS knowledge base and Ethernetted to a MicroVAX II running DECNet, FORTRAN and MSLiP under the Ultrix operating system.

2. An Apple MicroExplorer with 8mb of RAM and two 80mb hard disks and running the MacIntosh operating system, the MultiFinder multitasking environment, the Explorer operating system (on a separate Explorer board), DECNet, Common LISP, KEE and MIDAS. The MicroExplorer is also Ethernetted to the MicroVAX II.

3. For smaller optimization problems, it may be possible to run the system entirely on the MicroExplorer, using MacIntosh FORTRAN; however, this has not so far been tested.

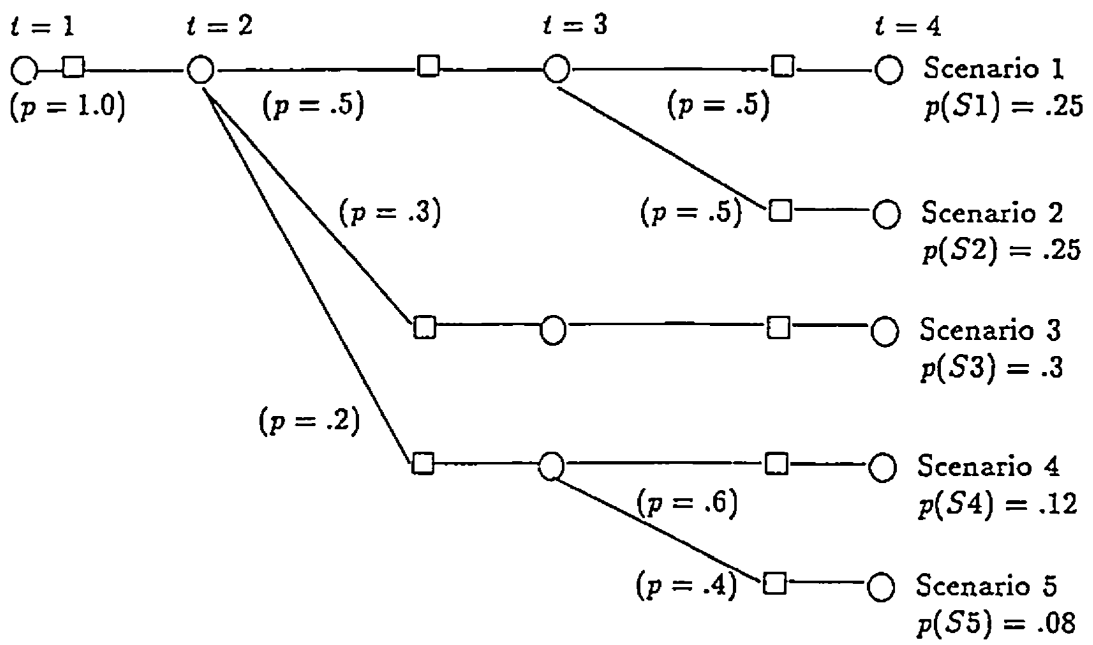
5
Model Specification

As outlined in Chapter 4, MIDAS' modelling subsystem carries out stochastic optimization, heuristic plan refinement and stochastic simulation of debt portfolios, integrated through a common underlying representation of domain and modelling knowledge. This chapter provides formal specifications for MIDAS' models and plan refinement rules; Chapter 6 will discuss their implementation and integration in the KEE knowledge base. Model validation is discussed in Chapter 9.

1. Rate event trees

As outlined in Chapter 3, future interest and exchange rates are the primary determinants of borrowing plan risk and return. Assumptions about these future rates are fundamental to all analysis and incorporated into the system as probability trees of significant rate movements over time (Figure 5.1), specified by the user. (In principle, any other uncertain parameters of the basic models, such as cash requirements, could also be included.) Each probability tree begins with rates at time $t = 1$ and branches when a significant economic or political event occurs which results in rate changes. At a branch, the conditional probability of the event's occurrence given prior events in the path is given by the user, together with new rates for each market affected by the event. (This is consistent in form with the data provided by external financial advisors.) Such *rate event trees* might incorporate either expert rate forecasts or subjective assessments of expected future rate movements; in either case, scenarios are specified as deviations from a base rate path as in Raiffa (1968) and Lane and Hutchinson (1980).

Each event sequence, denoted by (e_j) in the model specifications in this dissertation, may be thought of as a path through the rate event tree; the probability of



Notes

○	denotes a rate event.
□	denotes a set of decisions. There are no decisions in the final period; see Figure 5.3 for details.
p	denotes the conditional probability of a rate event given prior events.

$p(Sn), n = 1, \dots, 5$ denotes the scenario (path) probability.

Figure 5.1. Branching Rate Event Probability Tree

any single scenario is computed as the product of the conditional probabilities of the events on the path, and a degenerate rate event tree consisting of a single path gives the deterministic formulation of the borrowing problem.

The generic model is designed to be used over a *rolling* time horizon. When the planning period is advanced one time period, unrealized first period branches are pruned from the tree, realized beginning rates are stored and any necessary changes to subsequent events are made before the model is rerun for the new period.

1.1 Rates in the optimization model

Each decision variable and stochastic coefficient in MIDAS' optimization model is specified with respect to a rate event sequence (e_j) with probability $p(e_j)$. Each (e_j) corresponds to a scenario in the user-specified rate event tree which specifies a sequence of future expected (mean) rates by time period based on the events in the scenario. As described in detail below, the model optimizes over the entire tree, given the probability of each event sequence (e_j).

1.2 Rates in the simulation and projection models

For MIDAS' simulation modelling, random future interest and exchange rates are generated from distributions consistent with the underlying rate event trees.

Interest rates for debts, sinking funds and short-term investments specified in the borrowing plan are determined from randomly-generated *yield curves* (equations expressing yield to maturity as a function of term) in each financial market in which new debts are issued. A base yield curve is first generated for each financial market for each interval over the planning horizon. This is done by (a) randomly generating enough points to determine a curve and (b) connecting the points with a function approximating a yield curve shape. Because historical rate data is most readily available for government securities, MIDAS' base yield curves reflect government rates. Yields for specific debt and investment types for the borrower are estimated by adding spreads to the rate taken from the base yield curve. Spreads may be estimated from historical averages or generated randomly based on historical distributions; in the current implementation, a single mean spread estimate is used in all time periods for each debt type.

At present MIDAS uses a simple logarithmic curve form based on two points— a long-term rate (25 or 30 years, depending on the availability of historic data on which to base the distributions) and a short-term rate (3 months or 1 year). Following Howard (1986), rate changes are assumed to follow a random walk within specified lower and upper bounds.

New rates in a given market are generated as follows: at any time t and given a prior event sequence (e_j), $j = 1, \ldots, t-1$, the long-term rate at time t, denoted by $\mathrm{l}_t(e_j)$, is generated according to:

$$\Delta \mathrm{l}_t(e_j) := \mathrm{l}_t(e_j) - \mathrm{l}_{t-1}(e_j) = \mathrm{r}_t(e_j) \tag{5.1}$$

where $\mathrm{r}_t(e_j)$ is a random variable with a normal distribution (Ayers and Barry 1979) with mean, specified earlier through the rate event tree, depending on t and (e_j). The short-term rate at time t, for the event sequence (e_j), is given by

$$\Delta \mathrm{s}_t(e_j) := \mathrm{s}_t(e_j) - \mathrm{s}_{t-1}(e_j) = w\mathrm{r}_t(e_j) + \mathrm{v} \tag{5.2}$$

where ω is a constant and v is a random variable with a normal distribution with mean 0 and variance based on historic rates in the relevant financial market. The yield curve connecting the short- and long-term rates at time t has the form

$$y = a + b\ln(\tau) \tag{5.3}$$

where y is the yield and τ is the term (time to maturity) of the debt. The exact curve is, of course, determined once the two points are generated, and the current rate scenario or event sequence (e_j) for $j = 1, \ldots, T$ is determined by the long-term and short-term rate movements realized through this random rate generation.

The yield curve form used here, based on Bradley and Crane (1975), was chosen as a useful first approximation of yield curve shape. More complex forms, based on three or more points, have been established to give better fits to past data; for example, Bradley and Crane (1975) use a three-point exponential specification and Fooladi and Roberts (1990) use a polynomial of fifth degree or less. Current simulation models used for option pricing (Adamidou *et al.* 1989, Asay *et al.* 1989, Zenios 1989) suggest several other rate generation and curve- fitting methods (Vasicek and Fong 1982, Black *et al.* 1987). Inclusion of one or more of these techniques would be a natural extension to improve the precision of MIDAS' simulation.

Foreign exchange rates in each market are generated by a separate random-walk model. Due to lack of data, interest and exchange rates are simulated independently at this time; simulation realism would be improved by models which incorporate some form of interest rate parity partially linking movements in the two rates.

For cash flow projection, yield-curve sequences are calculated using (5.3) with the mean short- and long-term rates found in the rate event tree. Foreign exchange rates are taken directly from the rate event tree.

Full specification for the system's interest and exchange rate generation models is given in Appendix A.

2. The optimization model

The optimization feature in MIDAS applies scenario-based dynamic stochastic linear programming to long-term borrowing with a variety of debt markets and features. The debt portfolio model used in MIDAS extends the deterministic and stochastic portfolio models of Bradley and Crane (1972, 1973, 1975, 1980); Crane *et al.* (1977); Lane and Hutchinson (1980) and Shapiro (1987). These models have several common characteristics. First, each of them builds an optimal portfolio from a set of predefined available debt or investment alternatives. Second, all except Crane *et al.* (1977) directly incorporate uncertainty through branching probabilistic rate scenarios. Third, all produce borrowing or investment plans stated as series of decisions to borrow, hold or repay debt (or buy, hold or sell investments); in the stochastic models, these decisions are contingent on prior rate movements and, of course, on prior decisions.

As noted in Chapter 3, objective functions and constraints vary among models. Bradley and Crane maximize expected ending portfolio value subject to cash availability and maximum loss constraints; Lane and Hutchinson maximize expected total return subject to various probabilistic constraints which limit risk; and Shapiro minimizes discounted total purchase cost subject to constraints on portfolio attributes and maximum future required contributions. Crane *et al.* (1977) consider a short-term debt portfolio using deterministic linear programming; they minimize discounted total costs including ending discount or premium, subject to maturity mix and market constraints.

The specification of the generic MIDAS optimization model, formally presented in Figure 5.2, draws some features from each of these models and extends the approach to long-term borrowing over planning periods of arbitrary length (within solvability limitations). Each instantiation of the model produces a set of contingent borrowing decisions, based on user-specified available debt types, which minimize the expected ending value of the resulting debt portfolio at the user's planning horizon, given borrowing requirements, operating and marketing constraints and probability trees of future rate movements. The following chapter subsections describe the model in detail.

2.1 Subscripts and superscripts

Subscripts s and t refer to time periods, quarterly or annual at the option of the user; T is the total number of time periods in the planning period.

Superscript k identifies debt types available to the borrower, distinguished by market of issue (domestic or foreign, public, private or consumer), term, the presence or absence of a call option, the presence or absence of a sinking tı nd provision, and the presence or absence of a redemption provision, to model consumer savings bonds. Each debt outstanding at the start of the modelling period is represented by an individual debt type, and the set of debt types to be considered for new borrowing in a single model instantiation is determined by the user, selecting from the debt classes and features represented in the system's knowledge base.

2.2 Decision variables

The model's decision variables are the *amounts* of each debt type *borrowed*, *outstanding* and *retired* in each time period and the amount of *surplus cash* held throughout each time period. The surplus cash and short-term borrowing variables allow for

MIDAS STOCHASTIC PROGRAMMING MODEL SPECIFICATION

Notation

$s, t = 0, \ldots, T$ denote *time periods*

T is the *length* of planning period or *horizon*

$k = 1, \ldots, K$ denotes an available *debt type*

$e_j := e_{j1}, e_{j2}, \ldots, e_{jT}$, $j = 1, \ldots, J$ denotes a *sequence* of (rate) *events*.

(e_j) indicates that a variable or parameter is *contingent* on the event sequence e_j.

Decision variables

$B_t^k(e_j)$	dollar amount at par of debt type k *borrowed* at the beginning of period t.
$O_{s,t}^k(e_j)$	dollar amount at par of debt type k borrowed in period s and *outstanding* at the beginning of period t.
$R_{s,t}^k(e_j)$	dollar amount at par of debt type k borrowed in period s and *retired* at the beginning of period t.
$S_t(e_j)$	dollar value of *surplus* cash held throughout period t.

Parameters

$r_{s,t-1}^k(e_j)$	*interest paid* in period t per dollar of debt type k borrowed in period s and *outstanding* at the beginning of period t.
$c_{s,t}^k$	*sinking fund contributions* in period t per dollar of debt type k borrowed in period s and *outstanding* at the beginning of period t.
f_t^k	*issue costs* (excluding premium or discount) per dollar of debt type k borrowed in period t.
$g_{s,t}^k(e_j)$	*cash outflows* per dollar for debt type k borrowed in period s, if *retired* at the beginning of period t. (These parameters are used to define *call options*, handle sinking fund withdrawals and value the debt portfolio at the end of the planning period.)
$i_t(e_j)$	*interest rate* per dollar applicable to surplus cash held throughout period t.
$\rho_t^k(e_j)$	*exchange rate* of foreign currency per unit of base currency appropriate to debt type k in period t.
$p(e_j)$	*probability* of event sequence e_j, $j = 1, \ldots, J$. ($\sum_{j=1}^{J} p(e_j) = 1.$)
C_t	*cash requirement* for period t. If negative, C_t indicates an *operating surplus*.

M_t	*maximum* allowable *cash outflows for debt service* in period t.
N_t	*maximum total borrowing* over all debt types in period t.
q_t^k	*minimum borrowing* of debt type k in period t.
Q_t^k	*maximum borrowing* of debt type k in period t.
$L_t(e_j)$	*minimum* dollar amount of debt (at par) *retired* in period t.
$U_t(e_j)$	*maximum* dollar amount of debt (at par) *retired* in period t.
$O_{0,1}^k$	*dollar amount* of debt type k outstanding at the *beginning* of period 1.
S_0	*initial cash* surplus.

Objective

$$\begin{aligned}
&\min E(\mathbf{D}_T) \\
&= \sum_{j=1}^{J} p(e_j)\left\{\sum_{k=1}^{K} \rho_{T+1}^k(e_j) \sum_{s=0}^{T} g_{s,T+1}^k(e_j) O_{s,T+1}^k(e_j)\right. \\
&\qquad \left. -[1 + i_T(e_j)]S_T(e_j)\right\}
\end{aligned}$$

(expected cash outflows required to retire outstanding debt at the *end* of period T).

Constraints

Cash Requirements

For $j = 1,\ldots,J$ and $t = 1,\ldots,T$

$$\begin{aligned}
C_t = &\sum_{k=1}^{K} \rho_t^k(e_j)\left\{(1 - f_t^k)B_t^k(e_j)\right. && \text{(net new borrowing)} \\
&- \sum_{s=0}^{t-1}\left[\left(r_{s,t-1}^k(e_j) + c_{s,t}^k(e_j)\right)O_{s,t}^k(e_j)\right. && \text{(cash outflows to service outstanding debt)} \\
&\left.\left. + g_{s,t}^k(e_j)R_{s,t}^k(e_j)\right]\right\} && \text{(cash outflows on retirement)} \\
&+ S_{t-1}(e_j) && \text{(surplus cash at the end of the previous period)} \\
&+ i_{t-1}(e_j)S_{t-1}(e_j) && \text{(interest earned on surplus cash)} \\
&- S_t(e_j) && \text{(surplus cash at the end of current period).}
\end{aligned}$$

Debt inventory by type

For $j = 1,\ldots,J,\ s = 0,\ldots,t-2,\ t = 2,\ldots,T+1$ and $k = 1,\ldots,K$

$$\begin{aligned} O^k_{s,t}(e_j) - O^k_{s,t-1}(e_j) + R^k_{s,t-1}(e_j) &= 0 \\ O^k_{t-1,t}(e_j) \qquad\qquad - B^k_{t-1}(e_j) &= 0 . \end{aligned}$$

Maximum cash outflows for debt service

For $j = 1,\ldots,J$ and $t = 1,\ldots,T$

$$\textstyle\sum_{k=1}^{K} \rho^k_t(e_j) \sum_{s=0}^{t-1} (r^k_{s,t-1}(e_j) + c^k_{s,t}(e_j)) O^k_{s,t}(e_j) - i_{t-1}(e_j) S_{t-1}(e_j) \leq M_t .$$

Maximum total borrowing

For $j = 1,\ldots,J$ and $t = 1,\ldots,T$

$$\textstyle\sum_{k=1}^{K} \rho^k_t(e_j) B^k_t(e_j) \leq N_t .$$

Maximum debt issue size

For $j = 1,\ldots,J,\ t = 1,\ldots,T$ and $k = 1,\ldots,K$

$$B^k_t(e_j) \leq Q^k_t .$$

Minimum debt issue size

For $j = 1,\ldots,J,\ t = 1,\ldots,T$ and $k = 1,\ldots,K$

$$\textit{either}\ \ B^k_t(e_j) = 0 \ \ \textit{or}\ \ B^k_t(e_j) \geq q^k_t\ (\geq 0).$$

Maturity smoothing

For $j = 1,\ldots,J$ and $t = 1,\ldots,T$

$$L_t(e_j) \leq \textstyle\sum_{k=1}^{K} \sum_{s=0}^{t-1} R^k_{s,t}(e_j) \leq U_t(e_j).$$

Nonnegativity

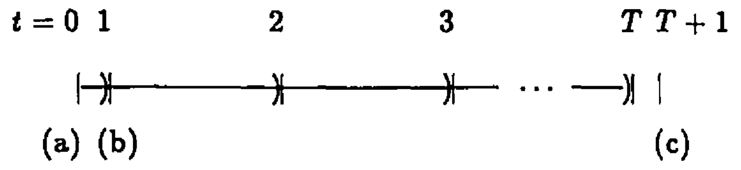
For $j = 1,\ldots,J,\ s = 0,\ldots,t-1,\ t = 1,\ldots,T$ and $k = 1,\ldots,K$

$$B^k_t(e_j) \geq 0 \qquad O^k_{s,t}(e_j) \geq 0 \qquad R^k_{s,t}(e_j) \geq 0 \qquad S_t(e_j) \geq 0.$$

For $j = 1,\ldots,J,\ s = 0,\ldots,T$ and $k = 1,\ldots,K$

$$O^k_{s,T+1}(e_j) \geq 0 .$$

Figure 5.2. Optimization Model Specification



Notes

(a) Opening debt balances are as at $t = 0$, immediately before $t = 1$.

(b) All debt transactions (issue, repayment, interest payment, calls, etc.) take place at $t = 1, 2, \ldots, T$.

Interest paid at each time t is calculated on principal outstanding after all transactions at time $t - 1$.

(c) Outstanding debts are valued at $t = T + 1$, immediately after all transactions at $t = T$.

Figure 5.3. Debt Planning Time Line

simple recourse through overborrowing and holding cash (invested at current short-term earnings rates) in some periods. Because interest and exchange rates determine costs for new and retiring debt, the decision variables and many of the parameters of the model are dependent on the prior rate movements (e_j) which, of course, determine current rates in any time period.

The model assumes *discrete* time periods (Figure 5.3) in which rates change instaneously at the beginning of each period (for period 1, rates are known) and all decisions for the period are then made instantaneously, again at the beginning of the period. Balances forward are calculated based on decisions in the previous period, and interest cost or earnings calculations are based on rates in effect in the previous period. Initial debt ($O^k_{0,1}$) and cash surplus (S_0) balances are outstanding at the start of the planning period. Note that these are initial fixed parameters shown in the constraints of Figure 5.2 as decision variables and therefore the maximum cash outflows in period one are predetermined. The final valuation of the debt portfolio takes place at the beginning of period $T + 1$, following the last rate change. No decisions are made in period $T + 1$.

2.3 Objective function

The model's objective function expresses the expected value of cash required to retire the total debt outstanding at market prices at the planning horizon (the end of the planning period). This approach eliminates the need to specify a corporate discount rate for measuring discounted costs, which would present a problem since the borrower's cost of new debt is determined by the solution to the model rather than exogenously.

The per-dollar ending value of each outstanding debt at the end of the modelling period is its theoretical market value, calculated as the net present value of future cash flows for the debt discounted at final-period market rates resulting from the rate event sequence. Consistent with market practices, callable bonds are valued using cash flows and market rates which assume that they are called at the first opportunity after the planning period ends; short-term debt, investments and sinking funds are valued at their principal outstanding at the planning horizon. For bonds issued during the planning period, coupon rates are assumed equal to market rates so that there is no premium or discount on issue. (Bond premium or discount values could be included as part of issue costs if rate models included parameters to separately estimate coupon rates and market rates; however, these have been eliminated to reduce complexity.) Interest rates for existing fixed-rate long-term debts are known at the start of the planning period.

2.4 Constraints

2.4.1 Cash requirements

The cash requirements constraint assumes that net requirements for new cash are exogenously specified from forecast operating and capital surpluses or deficits before interest payments and debt transaction costs. This is consistent with current operating practices at the Corporation, as noted in Chapter 3. Cash requirements can be negative in any time period, indicating a surplus available for debt repayment or investment. The constraint states that borrowing (net of issue costs) in each time period must equal the sum of the exogenously determined cash requirement for that period, interest payments on all outstanding debt in that period, sinking fund payments where required, cash outflows on debt repayment in that period (including retirements and calls) and surplus cash held in that period, net of the prior period's surplus cash and interest earned on that cash. All cash inflows and outflows for foreign-currency denominated debt are adjusted by current foreign exchange rates.

This constraint requires that all interest payments and transaction costs be financed by debt or by cash surpluses, so that the ending portfolio value reflects the cumulative impact of all cash requirements and borrowing decisions during the planning period.

2.4.2 Debt inventory constraints

An inventory constraint for each debt type in each period maintains consistency among amounts issued, outstanding and retired. Debt cannot be issued and retired in the *same* period. The inventory constraints therefore state that the outstanding amount in a period t of debt issued in period $t-1$ equals the amount issued in period $t-1$; the outstanding amount in a period t of debt issued in a period prior to period $t-1$ equals the amount of that debt outstanding in period $t-1$ less the amount of that debt retired in period $t-1$.

2.4.3 Maximum cash outflows for debt service

Cash outflows for debt service include interest and sinking fund payments required while debt is outstanding.

These constraints place a dollar limit on debt service cash outflows in each period regardless of rate scenario. The restriction incorporates into the model an expression of aversion to downside risk which remains computationally tractable, partially compensates for the risk-neutral objective function and appears to correspond to the behaviour of corporate debt managers (Bradley and Crane 1975).

2.4.4 Maximum total borrowing

These constraints limit total borrowing in any single time period, preventing run-away borrowing in periods with very low interest rates. The constraints reflect market ability to absorb only a limited amount of debt at one time and also incorporate aversion to the risk inherent in satisfying all cash requirements by borrowing in a single period or a few time periods.

2.4.5 Maximum and minimum debt issue size

Constraints on maximum and minimum amounts to be borrowed from individual sources in any one period can be determined within or outside the borrowing firm. Market limitations on the amount of debt that can be absorbed at one time or on the minimum practical size of an issue are externally-determined constraints of this type; internally-determined maxima and minima arise from the user's operating policies.

Short-term debt such as bank credit is retired and refinanced in the model at the beginning of *each* time period. For these debts, the amount borrowed in a period equals the amount outstanding in the period, so that the maximum issue size constraints also represent limits on total credit.

2.4.6 Maturity smoothing

These constraints reflect borrower policies limiting the amount of retired debt in any one period.

2.4.7 Nonnegativity

These constraints limit the decision variables to nonnegative values, with the results that all cash deficits are covered by borrowing and all cash surpluses show in the cash surplus variable.

2.5 Optimization model structure

This formulation of the strategic debt planning problem is technically a scenario-based *dynamic stochastic recourse model.* Dempster (1988) surveys the theoretical aspects of such models based on general discrete time stochastic processes of data evolution. A standard input format for these problems has been specified by Birge *et al.* (1987). (MIDAS's stochastic programming model input is implemented to this FORTRAN standard, so that the data defining a model instance is input to the solver from the MIDAS knowledge base in this format, and translated by internal FORTRAN subroutines to a data structure appropriate for the solver used. Details are given in Chapter 6.)

The problem's matrix structure can best be understood by first considering a *single-scenario model* and subsequently extending it to a two-period stochastic model and then to an arbitrary number of additional periods. A single-scenario (deterministic), three-period model has the form

$$A_1x_1 + A_2x_2 + A_3x_3 = b,$$

where A_1, A_2 and A_3 are the structured, i.e. largely sparse, constraint matrices for decisions x_1 in period 1, x_2 in period 2 and x_3 in period 3 respectively. More precisely, this model possesses *staircase structure* in which the constraints have the lower bidiagonal structure of discrete time control models. (The cash requirements and inventory constraints include data and decision variables from adjacent periods, while all other constraints in a given period are instantaneous, i.e. depend only on data and decision variables in that period.) This model's matrix structure is shown in Figure 5.4.

The stochastic form of this model is

$$A_1x_1 + \boldsymbol{A}_2\boldsymbol{x}_2 + \boldsymbol{A}_3\boldsymbol{x}_3 = \boldsymbol{b},$$

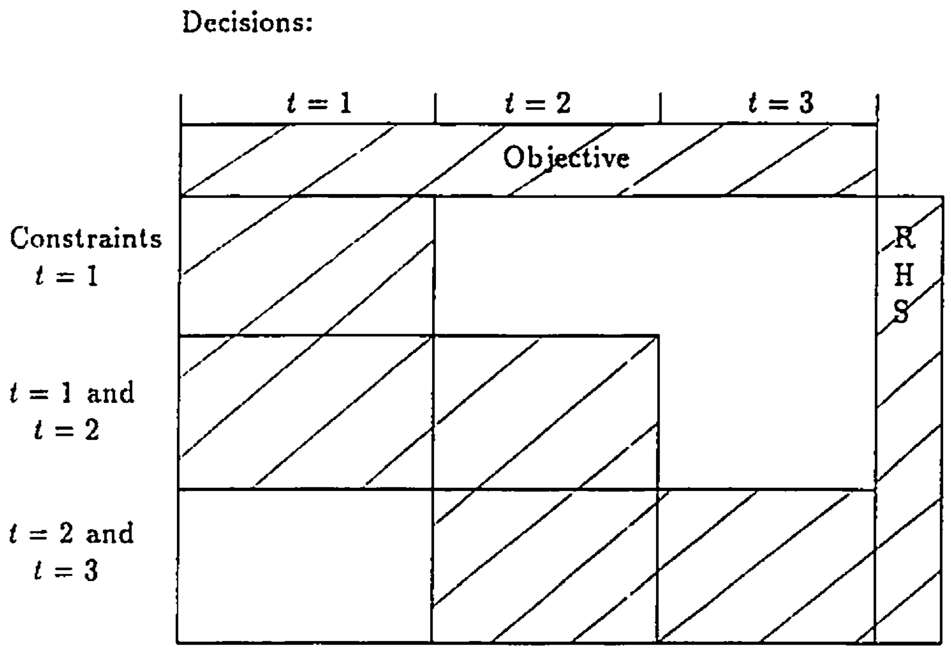


Figure 5.4. Three-period Deterministic Optimization Model Structure

where the cash requirement and inventory constraint portions of the matrices A_2 and A_3 and possibly some of the values in b are stochastic, i.e. rate scenario dependent. For the three-scenario tree shown in Figure 5.1, this matrix structure appears as in Figure 5.5. The matrix is constructed by scenario, working downward through the rate tree; the upper left-hand corner of the matrix specifies the first scenario, the structure of which duplicates that of the deterministic three-period example. Subsequent scenarios are added in path number order, each linked via its constraints to the decisions in the previous time period in the scenario from which it branches.

2.6 Solution technique

The stochastic nature of the recourse coefficients in this model reduces the efficiency of certain linear programming decomposition techniques, since each single-period, single-scenario subproblem can be unique in both its coefficients and its right-hand-

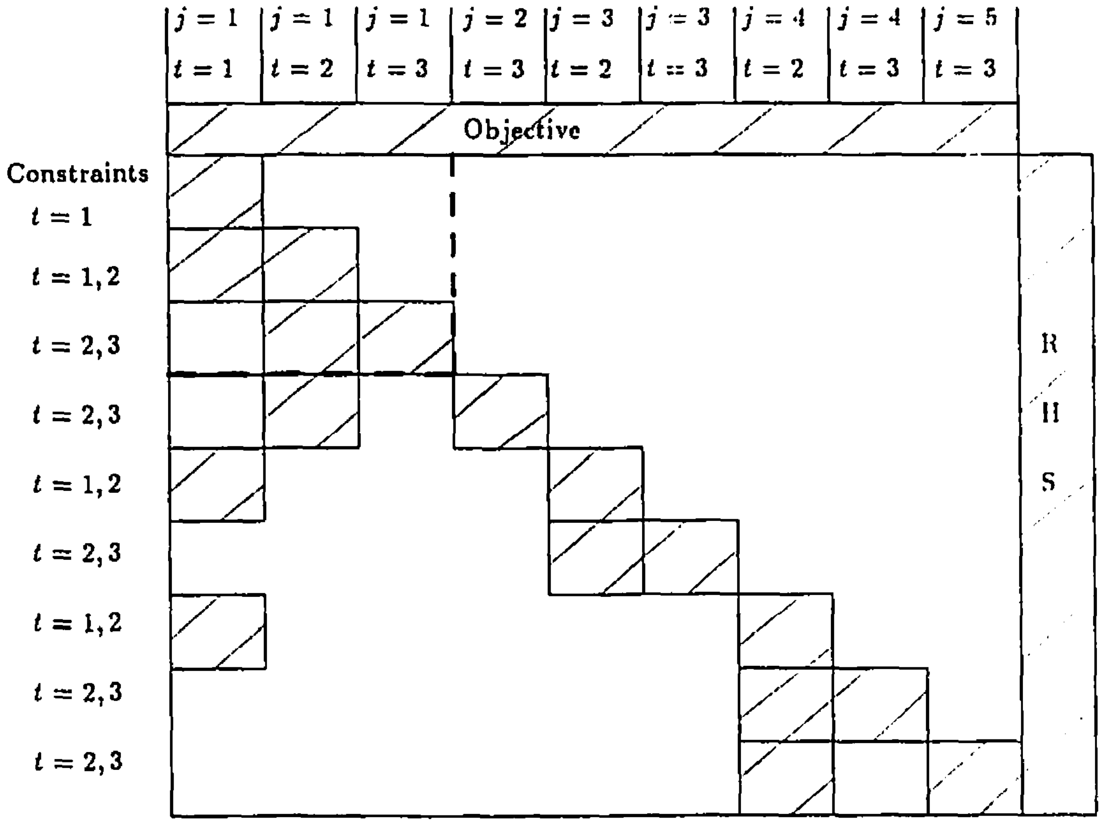


Figure 5.5. Three-period Stochastic Optimization Model Structure

side vector. Nested Benders decomposition (Benders 1962, Van Slyke and Wets 1969, Birge 1985, Gassmann 1989*a*) is the technique incorporated into the MSLiP code used in MIDAS for the solution of debt portfolio stochastic programming models. For this technique, the original problem is partitioned into a set of smaller linear programs, each of which represents only a single time period in a single scenario. The period one subproblem is solved first and its optimal solution passed to the next period; the process is repeated for each subproblem along each scenario. Conditions which lead to infeasibility or suboptimality in subproblems are noted and passed back along the scenarios as additional constraints (*cuts*), and the subproblems are re-solved. The entire process is iterated until no new information is generated. This technique has been shown to converge rapidly to an optimal solution (Birge 1985;

Gassmann 1987, 1989*a*,*b*). Gassmann has developed an efficient implementation of nested Benders decomposition which is used in the MSLiP program as the MIDAS stochastic programming model solver.

2.7 Model sizes and solution times

Individual optimization model size varies greatly with the formulation of the problem under consideration, specifically with (a) the number of markets, (b) the number of debt types, (c) the number of time periods, (d) the 'bushiness' of the rate event tree and (e) the types of constraint which are included in the problem statement. For example, a relatively small problem consisting of four scenarios branching at period 2, one market, five debt types and five time periods would have 1,599 decision variables and 826 constraints; expanding this problem to five markets and 20 periods would give over 150,000 decision variables and over 77,000 constraints. Test problems now being used with the solver have eight periods, two markets and with five debt types each and two credit lines, with four scenarios branching once at period 2; these have on the order of 1,900 decision variables and 1,600 constraints after elimination of irrelevant variables and constraints. These models are solved by MINOS on a MicroVAX II in 12 seconds. MSLiP is used for more 'bushy' problems; it solves problems with 20,000 decision variables and constraints in tens of seconds on a DEC 5000 Workstation.

2.8 Parametric analysis and EVPI analysis

Following the initial solution of an optimization model, it is desirable to rerun it a number of times in order to carry out parametric analysis on the key maximum cost constraint (and, if desired by the user, on other constraints). Such parametric analysis produces a curve relating changing constraint values to resulting new objective values for the optimal portfolio; for the maximum debt service constraint, this curve describes the relationship between debt service requirements (downside risk for the problem) and expected ending debt value.

In cases where the initial optimization model is too large to be easily rerun many times, analysis of the *expected value of perfect information* (EVPI) is used to prune the rate event tree to produce a smaller problem without reducing the analytical effectiveness of the model. For each node in the rate event tree, EVPI analysis measures the value of explicit foreknowledge of each branch from that node and collapses branches for which such foreknowledge would contribute little to changing the model's expected overall solution value. The analysis proceeds as follows:

1. For a given node (branch point) in the rate event tree, the optimization model is explicitly solved for each rate event path from that node to the end of the planning period. This produces the objective function value and decision variable values which would result if rates along this path were known to hold with certainty.
2. The EVPI for that node is computed as the expected value of the remaining dynamic stochastic optimization problem from the node, i.e. the optimal value of the objective function value produced by solving the stochastic problem (the problem considering all rate event tree branches) from that node to the end of the planning period, minus the probability-weighted sum of these objective function values.
3. If the difference between the EVPI is less than some prespecified cutoff value, the rate event tree branches from the node in question are collapsed into their probability-weighted average rate path, producing a smaller problem for further consideration.

The resulting smaller rate event tree is then used for reruns of the problem for parametric analysis purposes.

2.9 Optimization modelling's decision support role

Even after parametric analysis, optimization modelling does not completely solve the debt planning problem. It does produce an initial borrowing plan which is optimal, given assumed future rate scenarios, probabilities and specified available debt alternatives. However, these plans are not complete practical solutions to the problem because (a) they are oversimplified and (b) managers who do not have mathematical backgrounds tend not to trust complex optimization models to dictate decisions, as noted in Chapter 2. Thus the system allows for iterative reconfiguration of the models by interactively changing available debt types and parameters. The final optimization solution nevertheless only suggests a plan for further consideration.

3. Heuristic plan refinement

The task of converting optimization output into practical plans is a heuristic process which, in a strictly model-based system, would be carried out by a modelling expert. This expert would review optimization results with the Treasurer and modify them to produce practical recommendations.

This process is captured within MIDAS by a rule-based process which reviews optimization model output and modifies it using knowledge which would otherwise come from the (human) modelling expert and the Treasurer. Two types of modifications have initially been identified as necessary within MIDAS, although more are likely to be identified as the system is put into practical use; they are (a) conversion of minimum issue size constraints to integer values and (b) conversion of individual borrowing decisions to integer values.

3.1 Minimum issue size

The minimum issue size constraints stated in Figure 5.2 are not fully mathematically specified in the optimization model. They may be correctly handled using standard mixed integer linear programming techniques (see, for example, Shapiro 1987) with the following constraints for $j = 1,\ldots,J$, $k = 1,\ldots,K$ and $t = 1,\ldots,T$:

$$\begin{aligned} B_t^k(e_j) &- \delta_t^k(e_j)q_t^k \geq 0, \\ B_t^k(e_j) &- \delta_t^k(e_j)Q_t^k \leq 0, \end{aligned} \tag{5.4}$$

where $\delta_t^k(e_j) = 0$ or 1. To avoid computationally costly solution techniques including the heuristic tree search solution procedure used by Shapiro, MIDAS uses a relaxation of this last requirement to:

$$0 \leq \delta_t^k(e_j) \leq 1. \tag{5.5}$$

The constraint can then be handled by a linear program solver, but the decision values $\delta_t^k(e_j)$ must subsequently be set to 0 or 1 for a realistic plan. Plan refinement heuristics are used to check each issue against the market minimum and modify the constraint (5.5) to equal either 0 or 1. The optimization is then rerun to generate corresponding adjustments in other decisions.

3.2 Integer-valued borrowing decisions

The optimization model's formulation as a linear program enables the solution of large problems; however, it results in long-term borrowing decisions that are not practical because they are not in amounts that can in fact be issued in the markets specified. For example, a market in which bonds are generally issued in multiples of $10 million will not easily accept a decision to float a 25-year bond issue for $329.2 million. A second set of plan modification heuristics is used to check long-term borrowing decisions against market issue multiples and to modify the borrowing plan decisions for further testing through simulation and projection. (Modification of constraints would result in a fully specified solution for the optimization model prior to running the optimization solver.)

The representations and reasoning used for both types of plan refinement are described in Chapter 6. These heuristics are quite simple as they now stand, but they do perform key plan refinement functions. Additional modification needs will be identified as the system is used and plan refinement requirements are observed.

4. Simulation and projection modelling

MIDAS' simulation and cash flow projection facility performs the final analytical steps in the debt planning process. It carries out cash flow projections or stochastic simulation of full borrowing plans, parts of plans or individual debt alternatives, providing detailed information on the performance of alternative plans. This facility can be used either as a follow-up to the optimization and refinement processes just described, or it can be used alone to analyze debt plans or individual borrowing decisions proposed outside the system.

4.1 Simulation and projection applications to debt and portfolio management

Cash flow and profit projection is a widely-used approach to financial planning by corporate managers, as noted in Chapter 2; its extent has undoubtedly increased drama'.'cally with the introduction of spreadsheet software. As seen in Chapter 3, stochastic (Monte Carlo) simulation has so far been less popular among managers but is somewhat developed as a portfolio management tool.

Researchers have applied simulation and projection to both individual debts and to portfolios, as previously noted. For a single debt or a few debts the general approach is to define the debt(s) to be simulated, assume a future interest (and, if applicable, exchange) rate path over the planning horizon and calculate cash flows or costs for the debts for discrete time intervals up to that horizon, summarizing them in a measure such as net present cost or internal rate of return. For projection, the process is usually repeated with varying assumptions to test the sensitivity of the result to certain conditions and to provide insight into the behaviour of the simulated debts as rates change. Simulation automatically repeats the process many times with pseudo-randomly generated future rates until a distribution of results is produced (Howard 1986). The distributions of these random rates about specified mean rates in each period represent the probability structure of *intraperiod* rate fluctuations.

Portfolio simulations are generally aggregate models used to test predefined portfolio management strategies on fixed-income investment portfolios. These models take as input the proportions of the portfolio value in selected maturity categories and calculate the return (or cost, for debts) for assumed or pseudo-randomly generated future interest rates over the desired planning horizon.

The MIDAS simulation and projection facility (for simplicity, referred to hereafter as simulated modelling) models both individual debts and portfolios, building portfolio models as collections of individually-defined debt models. This gives significantly increased flexibility in that it allows testing of a number of debt features in addition to maturity, and it is computationally tractable because of advances in computing power since the time of the portfolio models cited.

4.2 Simulation model specification

MIDAS' simulation is a discrete interval oriented process. Given a planning horizon, contingent borrowing plan, existing debt portfolio (if applicable) and assumptions about future interest and exchange rate scenarios, it produces cash flows and costs for quarterly or yearly time intervals over the planning horizon as well as summary measures of performance such as net present cost. Used in the deterministic (projection) mode, it produces detailed projections of cash flows and debt performance; in the stochastic mode, it provides probability distributions for key performance indicators which can be used to assess plan cost and risk. It uses parameters and relationships consistent with those in the optimization and models in detail the debt types included in the optimization model.

Figures 5.6 and 5.7 summarize simulation inputs and outputs; a full formal specification of the model is given in Appendix B. With the exception of randomly generated interest and exchange rates, provided by the system according to the scheme outlined earlier, inputs are specified by the user.

Cash flow calculations are specific to each debt or investment type. For debts and sinking funds they include principal borrowed and retired; issue costs; interest; retirement costs and sinking fund contributions, earnings and withdrawals. All cash flows for foreign debts and investments are calculated in both foreign currency and in Canadian dollars at current exchange rates. Portfolio cash flows are simply line-by-line sums of the relevant individual debts.

SIMULATION MODEL INPUTS

A. Planning period specification (number of time periods, time period length, starting and ending dates).

B. Borrower parameters (cash requirements, desired maximum debt service requirements per time period, desired maximum total borrowing per time period, desired maturity smoothing).

C. Market parameters (for each market and rate scenario)
   - rate model parameters as listed in Appendix A
   - minimum and maximum issue sizes by debt type
   - issue cost rate by debt type

D. Future rate assumptions
   - rate event tree covering the planning period, specifying mean short-term and long-term government rates and foreign exchange rates
   - mean or pseudo-random market rates by debt type and time period, generated from the rate event tree by rate models in each market.

E. Descriptions of available debt types (debt class, market, term and features).

F. Descriptions of existing debts as at the start of the planning period.

G. A borrowing plan consisting of a set of borrowing, outstanding and retirement actions (specifying action date, amount and type of debt) to be carried out during the planning period.

Figure 5.6. Simulation Inputs

Portfolio performance indicators include the ending portfolio market value which is used as the objective function in the optimization, as well as period-by-period operating cash flows, the cash flow internal rate of return (where defined) and other measures identified as useful by Corporation staff.

4.3 Simulation modelling's decision support role

The simulation modelling facility can be used in a number of ways within the overall decision support framework. As a stand-alone facility, it models deterministically or stochastically the future performance of individual debt alternatives. This extension of the Corporation's existing spreadsheet modelling was a user priority, as noted in

SIMULATION MODEL OUTPUTS

A. A portfolio of debts, sinking funds and cash surplus reflecting the specified existing debts and results of borrowing plan actions.

B. Borrowing actions to cover cash deficits through penalty borrowing and to invest cash surpluses in short-term bank deposits

C. For cash flow projections based on single mean-rate scenarios:
   - cash flows by time period for single debts and the tested portfolio (principal borrowed, issue costs, interest paid, sinking fund contributions, sinking fund withdrawals, principal retired, retirement costs, total cash flows, total operating cash flows)
   - memorandum items by time period for single debts and the tested portfolio, as applicable (debt status (new, active, callable, mature or inactive), principal outstanding, accrued interest, sinking fund earnings, sinking fund balance)
   - ending value of single debts and the tested portfolio
   - the internal rate of return (if defined) for total cash flows for single debts and the tested portfolio
   - the net present value of total cash flows, discounted at the user-specified corporate discount rate
   - constraint violation warnings identifying time periods in which market or borrower constraints are violated by the borrowing plan.

D. For cash flow projections based on branching mean-rate event trees:
   - all of the items listed in (B) for each rate scenario in the rate event tree
   - expected values for all of the above over all scenarios in the rate event tree.

E. For intra-period stochastic simulation for a single rate scenario:
   - frequency distributions for the ending values of single debts and the tested portfolio
   - frequency distributions for the internal rate of return of total cash flows for single debts and the tested portfolio
   - frequency distributions for the net present values of total cash flows and of operating cash flows for single debts and the tested portfolio.

F. For intra-period stochastic simulation for a branching rate event tree:
   - frequency distributions for expected values over the full rate event tree, for the results listed in (D).

Figure 5.7. Simulation Outputs

Chapter 3.

When used for hierarchical planning, simulation complements optimization modelling by permitting further in-depth analysis of borrowing plans resulting from the optimization and refinement planning steps. As a result, the system's user can study detailed cash flows and risk measures for planning alternatives, developing insight into the implications of alternatives. This overcomes two of the main limitations of optimization for practical planning—the oversimplification of optimization results and the inability of managers to study in detail the reasons for optimization results.

5. Research contributions to debt modelling

Both the optimization and simulation model specifications presented here extend known model types to handle more complex and realistic situations than was previously possible. In addition, their complementary use is a new approach to model-based debt planning which exploits the strengths and minimizes the weaknesses of both modelling techniques. The hierarchical planning approach defined for this project appears to extend readily to other problem domains; this is a promising area for further work.

6 Modelling Subsystem Design

MIDAS' modelling flexibility and power arise in large part from the complementary use of several techniques to represent and manipulate domain knowledge, problem descriptions and modelling knowledge. This chapter describes the design of the modelling subsystem, explaining its ability to formulate and integrate optimization, simulation and projection models of a problem as well as to integrate multiple models and heuristic planning components. The description covers the design for some modelling features which have not been implemented; current implementation status is presented after the design details.

In this chapter and the next two chapters, system design is presented by subsystem or subtask in a top-down manner, progressing from requirements to the knowledge representation and control techniques used to carry out the required functions. First, detailed functional specifications are listed for the subsystem; second, knowledge requirements are identified; third, the representation of each required knowledge category is presented; fourth, control and reasoning strategies are described; fifth, the implementation status of the design is outlined; and finally, the research contributions, strengths and weaknesses of the design are discussed. Selected examples, together with object diagrams, inheritance hierarchy charts and method structure charts produced using the Excelerator CASE tool (Index Technology Corporation 1989), are used to illustrate the system description. A comprehensive illustration of system operation is presented in Chapter 10, and detailed system documentation is found in Appendices C and D.

1. Task and knowledge requirements

MIDAS' modelling subsystem performs all tasks required for the optimization, plan refinement, simulation and projection processes shown in Figures 4.1 and 4.2. Specifically, the subsystem carries out the following functions:

For an individual model (optimization, simulation, or projection):

1. Specify the model from a problem description.
2. Instantiate the model by determining all required parameter values.
3. Solve the model.
4. Represent its results in a way interpretable by other models and by other subsystems.
5. Perform multiple runs of the model in response to further analysis requests.

For heuristic plan refinement:

1. Apply the appropriate rules to plan representations.
2. Represent modified plans so that they can be further analyzed.
3. Reinstantiate and re-solve either model in response to requests from the refinement rules.

For all models and rule applications:

1. Integrate models and heuristics by representing output from each operation so that it can be readily used as input to others.
2. Maintain consistent model formulations for both models as a problem description changes.

These functions require the following categories of knowledge:

1. Domain concepts, facts and relationships, forming a problem description.
2. Specifications and procedures for mapping a problem description into either an optimization or simulation/projection model structure.

3. Specifications and procedures for mapping domain facts into model parameter values.
4. Calculations required for each model.
5. Model control requirements, or the sequence of calculations and operations needed to formulate, solve and represent the results of each model.
6. Heuristic plan refinement knowledge.
7. Correspondence between model or rule output and the problem description.
8. Correspondence between model and rule outputs and inputs for integration purposes.

Each category may contain numeric, symbolic, algorithmic and heuristic knowledge.

2. Knowledge representation

The above knowledge categories are represented and manipulated using a combination of frames, object-oriented programming and rules. Frames are used to represent and describe domain entities in terms of their attributes, while object-oriented programming methods implemented in LISP carry out necessary modelling operations and calculations once the problem description is known; the frames and LISP methods together thus provide the mappings between domain facts and model parameters needed to formulate and modify models as problem descriptions change. Rules encode the heuristic reasoning used for the intermediary function of inspecting and modifying plan output.

A common domain model organizes all knowledge in the modelling subsystem. Figure 6.1 shows the subsystem's major object categories, message flows and data flows. Three categories of objects are used. *Model objects* represent the domain objects—financial instruments and portfolios—which result from borrowing actions and which directly produce the financial results for which borrowing is carried out. *Model support objects* represent domain entities in the debt planning environment; they provide assumptions, constraints and descriptions of current conditions which

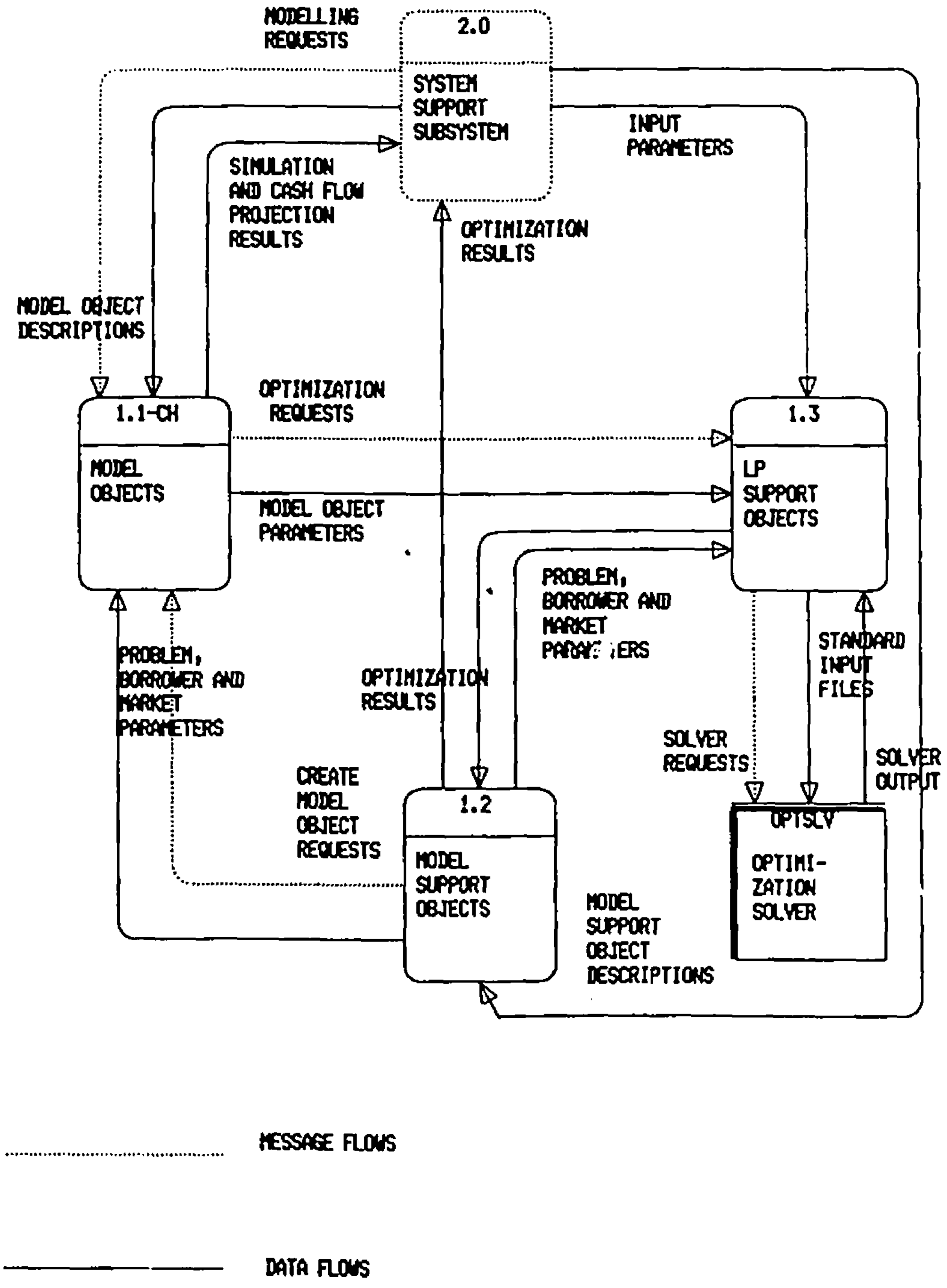


Figure 6.1. Modelling Subsystem

affect the development of future plans. *LP support objects* contain the conceptual and procedural knowledge needed to formulate appropriate stochastic programming models from domain descriptions and to translate optimization output into borrowing actions. Each object category consists of one or more KEE classes defined in terms of its attributes and procedures (methods); problems are described as collections of instances (individual members) of the classes.

2.1 Model objects

Figure 6.2 shows individual model object classes within MIDAS' modelling subsystem, together with their major data and message flows. Model objects are classified as either *basic* or *composite* objects, reflecting the system's portfolio-management view of the debt planning problem. Basic model objects describe and model individual debts and investments in a problem statement, functioning as building blocks for the system's models; composite model objects combine basic model objects into portfolios which control model processing and combine results for individual portfolio members.

Model objects are defined within the system using an inheritance hierarchy of portfolio, debt and investment classes (Figure 6.3). For financial instruments, the hierarchy follows the standard asset/liability classification scheme used by accountants and financial managers. The inheritance hierarchy streamlines model object definitions in several ways. First, attributes and their default values are defined at the highest possible level and inherited by subclasses and instances. Second, inheritance provides common method names such as 'do.cash.flows' for all model objects. Third, calculations associated with financial instrument classes are described by numerical algorithms, again defined at the highest possible level and inherited downward. And fourth, multiple inheritance is used to create instances of debts having features of more than one subclass, as when bonds with collections of features such as a call feature together with a sinking fund are required.

Financial instruments (basic model objects) include both debt and investment

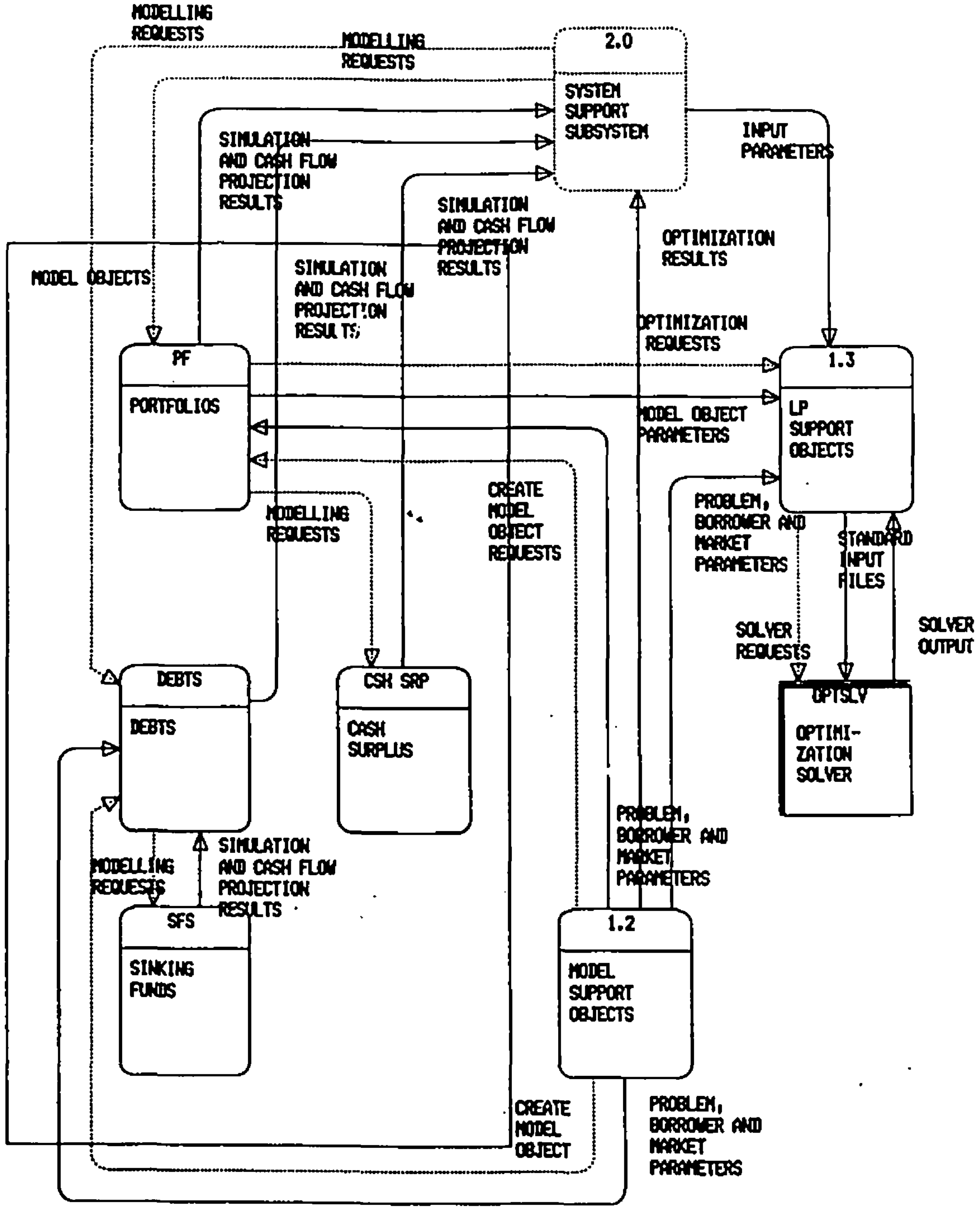


Figure 6.2. Modelling Subsystem Showing Model Object Detail

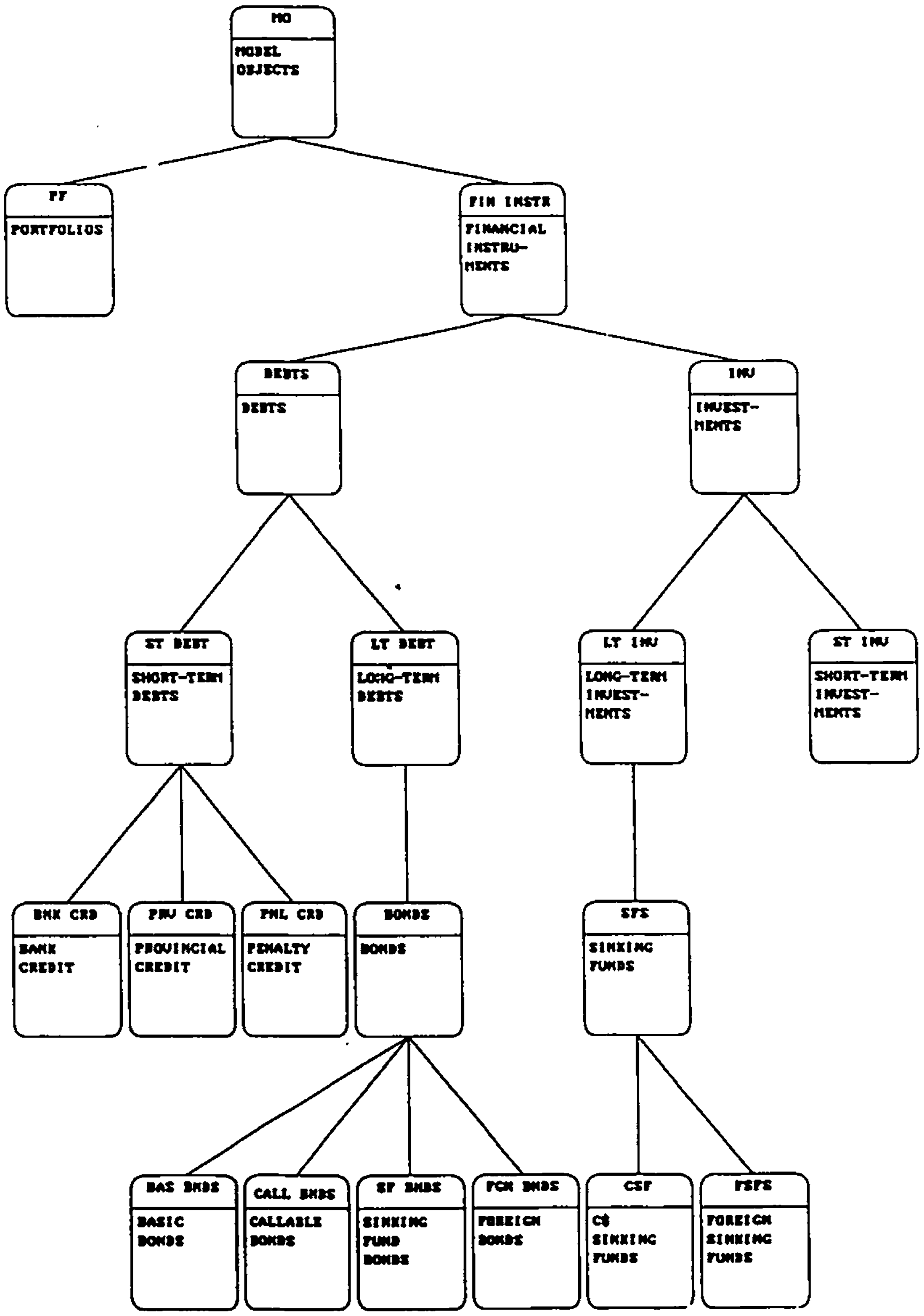


Figure 6.3. Model Object Class Hierarchy

NAME:	JJ
SUBCLASS.OF:	NIL
INSTANCE.OF:	CALLABLE.BONDS, SINKING.FUND.BONDS
ATTRIBUTE.SLOTS:	
accrued.interest	0.0
cf.table	JJ.TABLE
class	BOND
currency	C$
market	CANADA
principal.outstanding	50.0
term	25
original.principal	50.0
coupon.rate	10.5
interest.pmts.per.year	2
issue.date	6/30/1990
issue.total.costs	1.0
maturity.date	6/30/2015
quarters.interest.paid	2, 4
call.first.call.premium	.05
call.first.call.year	2010
call.part?	NIL
call.prices	(105.0 104.0 103.0 102.0 101.0 100.0)
sf.name	SF.JJ
EXTERNAL-USE METHODS:	
add	
change	
delete	
do.cfs	
project	
simulate	

Figure 6.4. A Basic Model Object

types which are relevant for debt planning. (The need for investments arises in debt planning when sinking funds are used and when surplus cash is invested for short periods.) Each financial instrument encapsulates descriptive attributes (interest rate, principal amount, issue date, term, maturity date, etc.) and their values. Each also contains LISP methods for model calculations and control (see further discussion below).

Figure 6.4 illustrates a typical debt object which has been formed using multiple inheritance. (This example and the others in this text show only definitional slots and top-level methods; the object as implemented contains additional slots and methods used in system operation. See Appendix C for full object definitions.)

NAME:	PF1
SUBCLASS.OF:	NIL
INSTANCE.OF:	PORTFOLIOS
ATTRIBUTE.SLOTS:	
class	PORTFOLIO
cf.table	PF1.TABLE
pf.markets	(CANADA US)
pf.members	(AA BB JJ X1 X2)
EXTERNAL-USE METHODS:	
add	
change	
delete	
do.cfs	
optimize	
project	
simulate	

Figure 6.5. A Composite Model Object

Portfolios (see Figure 6.5 for an example) are composite model objects which group individual debts and investments. Portfolio performance is modelled as the combined performance of the individual instruments in the group, so that financial instrument objects form submodels of comprehensive portfolio models.

Portfolios supervise optimization, simulation and projection for portfolios, handle interactions among objects and summarize portfolio results. It is these composite objects which provide the flexibility needed to configure models for debt portfolios reflecting alternative borrowing plans.

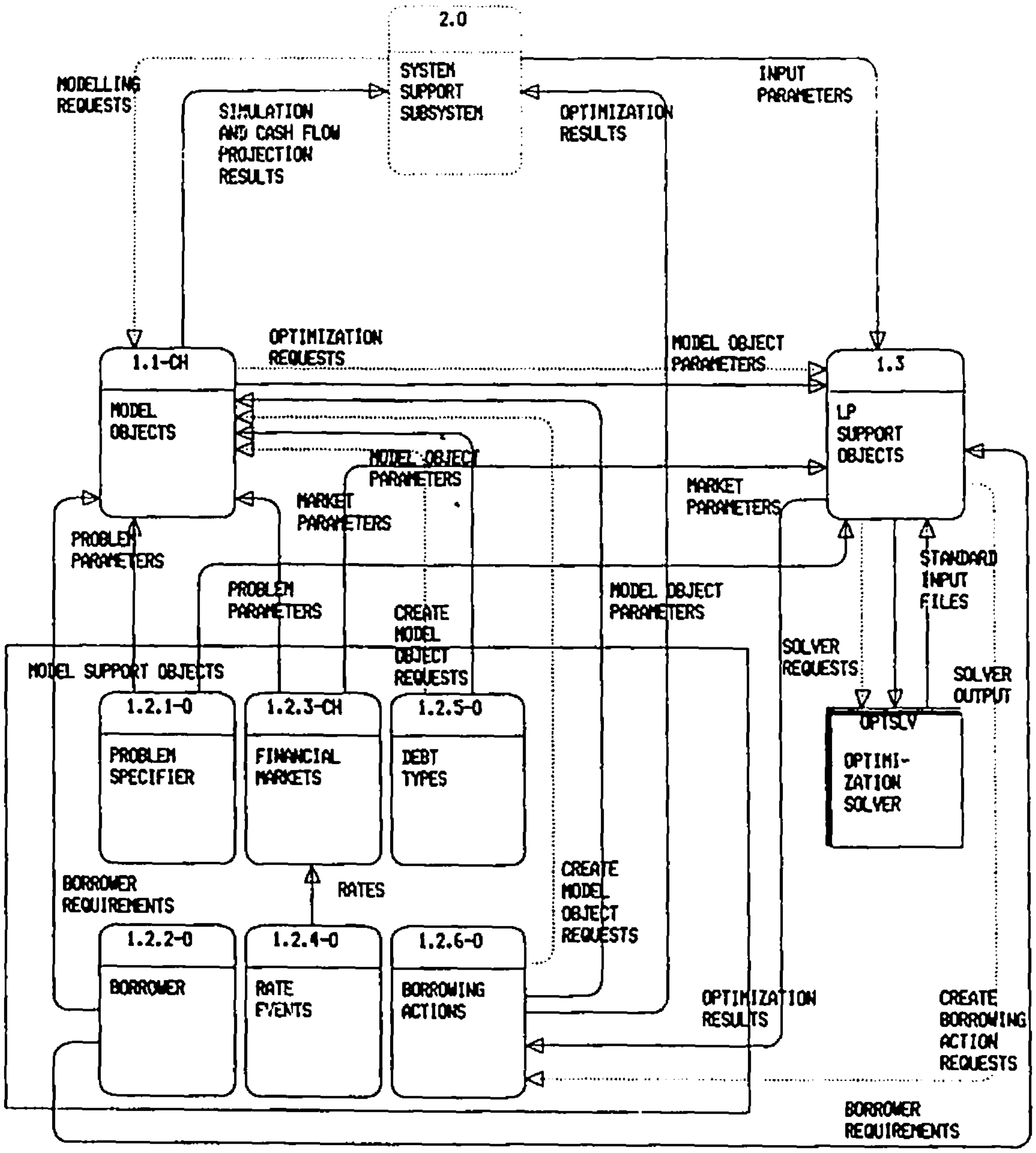


Figure 6.6. Modelling Subsystem Showing Model Support Object Detail

2.2 Model support objects

Model support objects include the problem specifier, borrower, financial markets (domestic and foreign), rate events, debt types and borrowing actions. Figure 6.6 shows model support object details in the context of the modelling subsystem.

The *problem specifier* holds overall problem parameters such as the length of the planning period, its starting and ending dates, the length of individual time periods (quarterly or yearly) and the *knowledge base date* (the starting date reflected in financial instrument balances, market conditions and rate assumptions in the knowledge base, which is used to check consistency among the starting date and opening rates and balances used in the models).

NAME:	NSPC
SUBCLASS.OF:	NIL
INSTANCE.OF:	BORROWERS
OWN SLOTS:	
cash.needs.actual.percent	.90
cash.needs.budget	((1990 1 50) (1990 3 25) (1991 2 100))
discount.rate	10.0
maximum.annual.borrowing	150.0
maximum.annual.debt.service	100.0
maximum.annual.retirement	100.0
minimum.annual.retirement	0.0
EXTERNAL-USE METHODS:	
add	
change	
delete	
cf.cash.needs	

Figure 6.7. A Borrower Object

The *borrower* handles financial goals, cash requirements and constraint parameters arising from borrower policies and preferences. A borrower example is shown in Figure 6.7.

NAME:	U.S.
SUBCLASS.OF:	NIL
INSTANCE.OF:	FOREIGN.MARKETS
ATTRIBUTE.SLOTS:	
cf.current.a.coeffs	(10.0 10.0 10.0 9.25 9.25)
cf.current.b.coeffs	(.588 .588 .588 .367 .367)
cf.current.rates.fx	(1.25 1.25 1.25 1.2 1.2)
cf.mean.rates.fx	(1.25 1.25 1.25 1.25 1.2)
cf.mean.rates.lt	(12.0 12.0 12.0 10.5 10.5)
cf.mean.rates.st	(10.0 10.0 10.0 9.25 9.25)
cf.rate.changes.fx	(0.0 0.0 0.0 -.05 0.0)
cf.rate.changes.lt	(0.0 0.0 0.0 -1.5 0.0)
cf.rate.changes.st	(0.0 0.0 0.0 -.75 0.0)
currency	US$
default.call.part?	NIL
default.int.pmts.per.year	2
default.issue.cost.rate	.05
default.bond.features	CALL, SF
default.call.wait.period	15
default.call.first.premium	.05
default.sf.contribution.rate	1.5
default.sf.wait.period	2
issue.multiple	10
maximum.issue.size	150
minimum.issue.size	25
quarter.alpha	1.0
quarter.sd.fx	.1
quarter.sd.lt	.25
quarter.sd.st	1.0
quarter.sd.st.random.part	.968
rate.event.slot	RATES.US
spread.bank.deposit	-.5
spread.basic.bond	.85
spread.call	.15
spread.penalty.credit	2.0
spread.prov.credit	-1.0
spread.sf	-.25
spread.sf.earnings	0.0
spread.yield.to.call	-.25
term.lt	30
term.st	1
year.alpha	1.0
year.sd.fx	1.0
year.sd.lt	1.05
year.sd.st	2.0
year.sd.st.random.part	1.73
EXTERNAL-USE METHODS:	
add	
change	
delete	
fx.rate	
interest.rate	
generate.mean.rates	
generate.random.rates	

Figure 6.8. A Financial Market Object

Financial markets (an example is shown in Figure 6.8) specify default debt characteristics for borrowing in various markets, hold market-related constraint values such as maximum allowable borrowing in a period and provide market interest and exchange rates (deterministic or stochastic) consistent with existing rate scenarios as required by mathematical models or rule sets.

The random interest and exchange rate generators described in Chapter 5 and Appendix A are implemented as methods in financial market objects. The general rate-generation algorithms, based on (pseudo) random number generation, are defined in the top-level market class and inherited by each market instance; equation coefficients and parameters, which customize each rate generator based on historical performance in its market, are stored as attribute values in the market instance.

```
NAME:           RET1.2
SUBCLASS.OF:    NIL
INSTANCE.OF:    RET

OWN SLOTS:
        event.description       Meech Lake rejected
        event.probability       .9
        event.quarter           2
        event.year              1990
        rates.canada            ((ST 14.5)(LT 12.25)(FX 1.00))
        rates.us                ((ST 10.0)(LT  9.0)(FX 1.25))
        tree.name               RET1

EXTERNAL-USE METHODS:
        add
        change
        delete
        set.up.scenarios
```

Figure 6.9. A Rate Event

Uncertainty is handled in MIDAS through *rate event trees*, which represent branching, probabilistic rate movements over the planning horizon, as described in Chapter 5. An individual *rate event* (Figure 6.9) contains a set of interest and

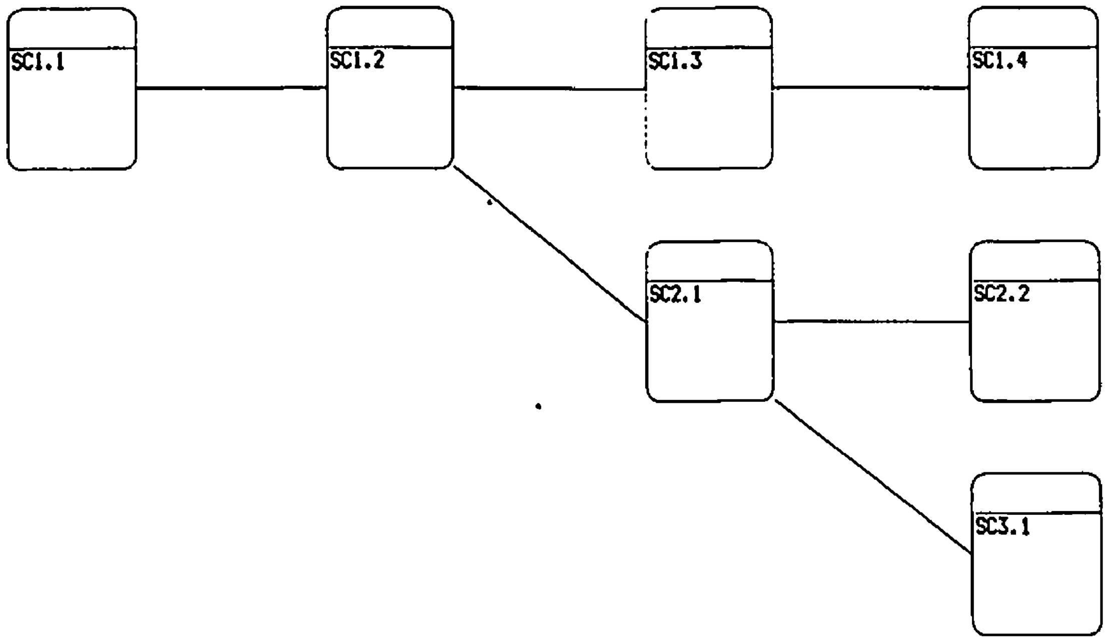


Figure 6.10. A Rate Event Tree

exchange rate realizations at a specified date, along with the event probability (conditional on past events) and an optional description of event causes. Rate event trees are formed by linking individual events into KEE inheritance hierarchies through subclass relationships as in Figure 6.10; a path through the tree defines a single scenario, and additional scenarios are formed by branching from higher paths as described in Birge *et al* (1987). (For an initial naming convention, rate events are named using event numbers following path numbers in the format TTP.E, where T is the tree identifier, P is the path identifier and E is the event number.) Each rate path is translated into a set of rate lists in financial markets at the start of problem analysis; these rate lists are the assumptions used for (a) generation of optimization coefficients, (b) mean-rate cash flow projections and (c) distribution means for random rate generation during simulation.

```
NAME:           C1
SUBCLASS.OF:    NIL
INSTANCE.OF:    DEBT.TYPES

OWN SLOTS:
    class         BOND
    description:  10-year Canada bond, noncallable.
    market        CANADA
    term          10
    features      NIL
    lp.name       C1

EXTERNAL-USE METHODS:
    create.hypothetical.portfolio
    select
```

Figure 6.11. A Debt Type

Debt types define the types of debt to be considered in developing a borrowing plan. A debt type is defined as a combination of debt class, such as bond, savings bond or short-term loan; market; term and debt features such as call option or sinking fund requirement. In addition to these defining parameters, each debt type contains methods for creating a portfolio of hypothetical debt objects corresponding to debts of the type issued in all type periods in a specified planning period. A debt type example is given in Figure 6.11.

```
NAME:            BC1.91
SUBCLASS.OF:     NIL
INSTANCE.OF:     BORROW.ACTIONS

ATTRIBUTE.SLOTS:
      action.date           2/1/1991
      action.quarter        1
      action.source         LP
      action.status         DONE
      action.year           1991
      amount                25.0
      debt.type             C1
      fi.name               C1.91
      lp.name               BC191
      constraints           (CR1 CR2 IV2 MB1)

EXTERNAL-USE METHODS:
      add
      change
      create.portfolio
      delete
```

Figure 6.12. A Borrowing Action

Borrowing actions specify the activities of borrowing, holding, and retiring debt and holding cash surplus to be carried out over the planning horizon. These actions are created to represent decision variables during the optimization modelling process. They are modified to reflect decision variable values following optimization and by plan refinement heuristics. Borrowing actions are also created and modified by the user to directly specify a plan for simulation or projection testing.

A borrowing action contains action specifications, its corresponding variable name in the optimization model, a list of optimization constraints in which it appears, and methods for creating and modifying a debt portfolio and individual debts. A borrowing action is shown in Figure 6.12.

2.3 Object maintenance

In addition to attribute values and model operators, all model objects and model support objects incorporate methods for handling their own maintenance through form-based input, consistency checks and attribute value displays. This allows system control methods to remain *general*, ignoring the maintenance details for individual submodels and support objects.

2.4 LP support objects

MIDAS' LP support objects and their interactions with the rest of the modelling subsystem are shown in Figure 6.13. These objects contain specifications and procedures for formulating, building input files for, solving and interpreting output from a stochastic programming model based on a given problem description stated as a domain model in the knowledge base, relying on explicit knowledge of stochastic model structure, input requirements, input sources, output structure and output destinations within the knowledge base.

LP structure specifiers (for decision variables, constraints and the objective function) are abstractions of model components which contain parameters and generalized methods for instantiating the components from a problem description. *Decision variable specifiers* are defined in a class hierarchy (Figure 6.14); each class contains specifications for a type of decision variable (borrowing action) to be created corresponding to existing and hypothetical future debts in a given problem. These objects use heuristics and knowledge of debts and market conditions to formulate a hypothetical borrowing plan as a customized *minimal* set of required decision variables for any problem description. Figure 6.15 shows a decision variable specifier example.

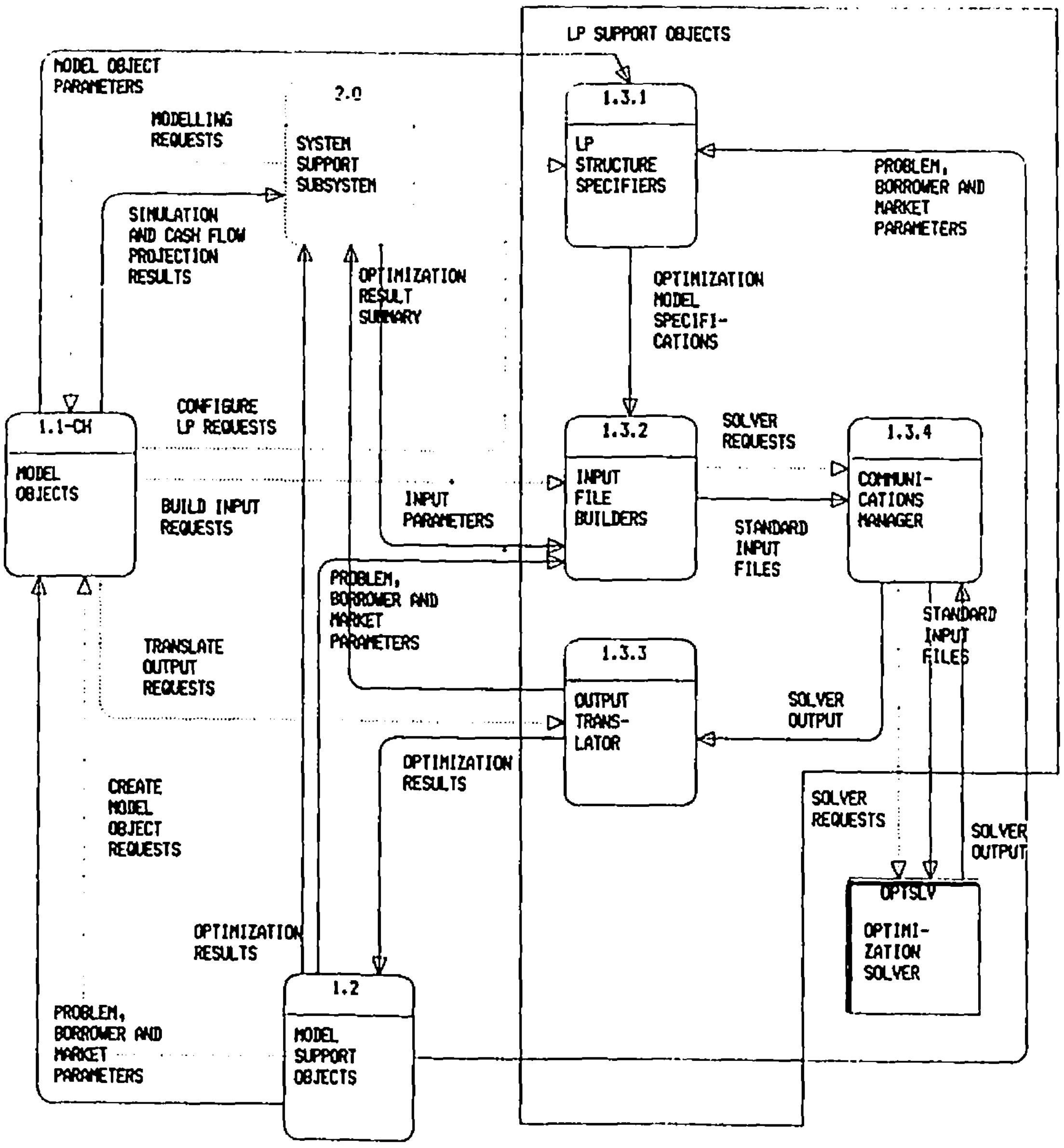


Figure 6.13. Modelling Subsystem Showing LP Support Object Detail

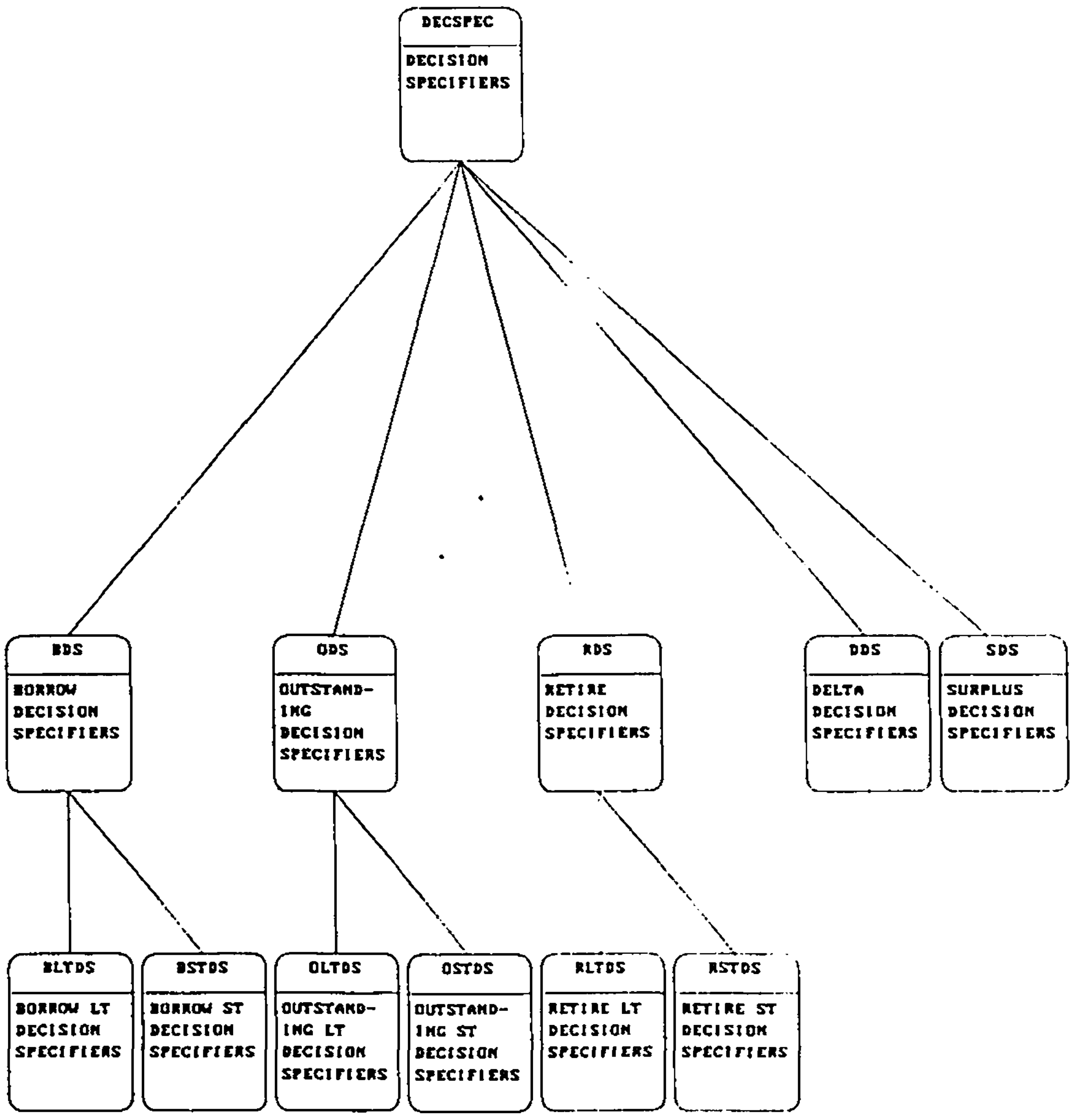

DECSPEC
DECISION SPECIFIERS
BDS
BORROW DECISION SPECIFIERS
ODS
OUTSTANDING DECISION SPECIFIERS
RDS
RETIRE DECISION SPECIFIERS
DDS
DELTA DECISION SPECIFIERS
SDS
SURPLUS DECISION SPECIFIERS
BLTDS
BORROW LT DECISION SPECIFIERS
BSTDS
BORROW ST DECISION SPECIFIERS
OLTDS
OUTSTANDING LT DECISION SPECIFIERS
OSTDS
OUTSTANDING ST DECISION SPECIFIERS
RLTDS
RETIRE LT DECISION SPECIFIERS
RSTDS
RETIRE ST DECISION SPECIFIERS

Figure 6.14. Decision Variable Specifier Class Hierarchy

NAME:	BORROW.LT.DECISION.SPECIFIER
SUBCLASS.OF:	BORROW.DECISION.SPECIFIERS
INSTANCE.OF:	
ATTRIBUTE SLOTS:	
decision.type	B
decision.period.maximum	(time period T)
name.format	(B + 2-digit.debt.type + 2-digit.decision.period)
description.format	(Borrow + decision.amount + 'of' + debt.type + 'in period' + decision.period)
object.classes	LONG-TERM DEBTS
decision.period.values	ISSUE.PERIOD
issue.period.source.slot	ISSUE.PERIOD
EXTERNAL-USE METHODS:	
create.plan	

Figure 6.15. A Decision Variable Specifier

Constraint specifiers and the *objective specifier* contain parameters and methods for data-driven constraint and objective function construction which are used to produce specific rows in the standard input files. (Constraints and the objective function are not explicitly represented as individual objects because within the present system design there is no need to refer to them except during input construction.) Constraint specifiers are defined in the class hierarchy shown in Figure 6.16. These specifiers state which decision variables are included in each input row together with sources or values for coefficients for each in the knowledge base and right-hand side sources or values. Figure 6.17 shows a constraint specifier object.

Input file builders contain specifications and procedures for constructing standard input files according to the format specified in (Birge *et al.* 1987) once model structure is known. There is an input builder for each of the standard input files, called the *core*, *time* and *stoch* files; using a detailed file outline and generalized methods, each input builder produces a file of FORTRAN card images to be read by the solver program.

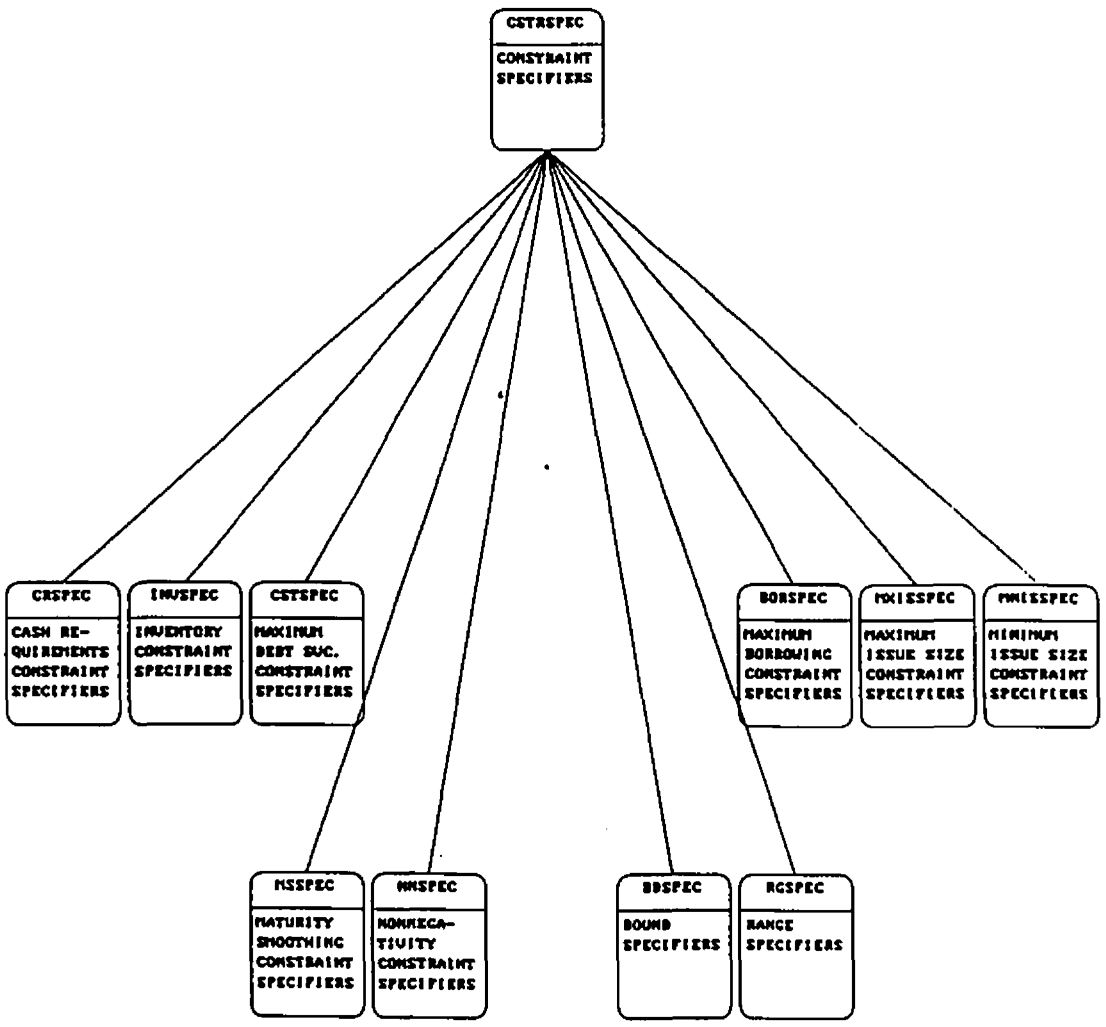

CSTRSPEC
CONSTRAINT SPECIFIERS
CRSPEC
CASH RE-QUIREMENTS CONSTRAINT SPECIFIERS
INVSPEC
INVENTORY CONSTRAINT SPECIFIERS
CSTSPEC
MAXIMUM DEBT SVC. CONSTRAINT SPECIFIERS
BORSPEC
MAXIMUM BORROWING CONSTRAINT SPECIFIERS
MXISSPEC
MAXIMUM ISSUE SIZE CONSTRAINT SPECIFIERS
MNISSPEC
MINIMUM ISSUE SIZE CONSTRAINT SPECIFIERS
MSSPEC
MATURITY SMOOTHING CONSTRAINT SPECIFIERS
NNSPEC
NONNEGA-TIVITY CONSTRAINT SPECIFIERS
BDSPEC
BOUND SPECIFIERS
RGSPEC
RANGE SPECIFIERS

Figure 6.16. Constraint Specifier Class Hierarchy

NAME:	MAX.ISSUE.SIZE
SUBCLASS.OF:	LP.CONSTRAINTS
INSTANCE.OF:	
DESCRIPTION:	

ATTRIBUTE SLOTS:

constraint.type	MX
translation.format	('Maximum issue size for' + debt.type)
decision.period.range	(1 through T)
issue.period.range	(1 through T)
sum.over.debt.types?	NIL
sum.over.issue.periods?	NIL
borrow.decisions	T
decision.period	t
coeff.value	+1.0
hold.decisions	NIL
retire.decisions	NIL
delta.decisions	NIL
surplus.decisions	NIL
equation.type	L
rhs.source.object.class	FINANCIAL.MARKETS
rhs.source.slots	MAX.ISSUE.SIZE
rhs.source.index	NIL
rhs.value	NIL

Figure 6.17. A Constraint Specifier

A *communications manager* contains parameters and methods to send the required input files to the solver, request problem solution and receive solver output. Communications parameters include pathnames for input and output files on both the Explorer and the solver machine, a solver machine login sequence, and a command sequence for executing a batch file to start the solver; methods are defined which log in and out of the solver, copy files in both directions, start the solver (which runs in either foreground or background, depending on user specification) and check for completion of the solver run. The solver works under the control of the communications manager, solving problems only at its request and returning FORTRAN-format output files containing values for all decision variables and objective and EVPI values for each node in the rate event tree.

Finally, an *output translator* reads the solver output file and updates borrowing plans and history records in the knowledge base to reflect relevant output values.

2.5 Rate scenarios and KEEWorlds

Branching rate scenarios are incorporated into both the optimization and simulation/projection modelling processes through KEEWorlds, which are the KEE implementation of worlds as described in Chapter 2 (Filman 1988). A KEEWorld can be thought of as a copy of the knowledge base in which all objects, inheritance hierarchies, attributes and attribute values are duplicated but in which slot values may be overridden. Each world thus has a unique set of facts which describe it, and a set of KEEWorlds for a given problem forms an inheritance hierarchy of worlds.

To implement branching rate scenarios in MIDAS, each full path in a rate event tree corresponds to a unique KEEWorld. This world consists of copies of all objects in the base world, giving an inherited problem description; however, portfolio members, future rate lists, corresponding borrowing actions and projected debt portfolio performance vary according to rate path. Expected values are computed in the top-level world, in which special objects and methods summarize results for all scenarios.

KEEWorlds are also used to represent multiple versions of a model during EVPI-based model modifications and reruns as explained in Chapter 5. The revised rate event tree resulting from EVPI analysis is represented as a new tree of rate event objects, which become the basis for a new set of worlds reflecting the new scenarios. The modelling process proceeds as for the initial model, with the exception that initial results can be compared to new results because both are available for analysis in their respective worlds.

MIS RULE 1:

```
If ?D is a delta.action
    And ?V is the value of ?D
    And 0 ≤ ?V < .5
Then ?V = 0 is a new constraint on ?D.
```

MIS RULE 2:

```
If ?D is a delta.action
    And ?V is the value of ?D
    And .5 ≤ ?V < 1.0
Then ?V = 1.0 is a new constraint on ?D.
```

Figure 6.18. Minimum Issue Size Rules

IVB RULE 1:

```
If ?B is a borrow.action
    And ?D is the debt object corresponding to ?B
    And ?M is the market for ?D
    And ?IM is the issue.multiple for ?M
    And ?A is the amount for ?B
Then ?A =?IM times the next integer greater than ?A/?IM.
```

Figure 6.19. Integer-valued Borrowing Decision Rule

2.6 Refinement heuristics

Initial specifications for heuristic plan refinement are presented in Chapter 5. Each of the two types of heuristics can be expressed as one or two rules, as shown in Figures 6.18 and 6.19. Although they could easily be coded as LISP procedures, they have been implemented using the KEE rule-based reasoning facility to illustrate its application for plan refinement and to incorporate a facility which can be expanded when additional refinement rules are identified.

The refinement rules access individual slot values in borrowing plan actions and financial markets; modifications are made by LISP expressions within the rules.

3. Reasoning and control

Using the knowledge representation just described, MIDAS is able to formulate, solve and modify both optimization and simulation/projection models. Each model is formulated by first building a model-object representation of the problem situation, including a borrowing plan, borrower and market conditions, rate scenarios and a hypothetical debt portfolio; this dynamic domain representation reflects the current state of problem analysis at any time and is referred to and modified by all modelling processes. For modelling, the representation's current attribute and cash flow values are translated into particular model formats and 'solved' using model operators (methods) specific to each model type.

3.1 Optimization modelling control

Figures 6.20 and 6.21 show the control structure for optimization modelling. (See Section 5 of Appendix C for a description of model structure chart conventions.) This processing takes place in the following steps:

1. Initialization

a. The user enters a problem description including a specified planning period, borrower cash requirements and constraints, market conditions, assumed future rate scenarios, existing debts and a set of possible debt types which can be used to meet the projected cash requirements.

b. A KEEWorld is created for each path in the assumed rate tree. (This step and those which follow are controlled by the 'optimize' method in the PORTFOLIOS class.)

c. A hypothetical debt portfolio is constructed which consists of all possible debts with principal amounts set at $1. This portfolio is inherited by all worlds.

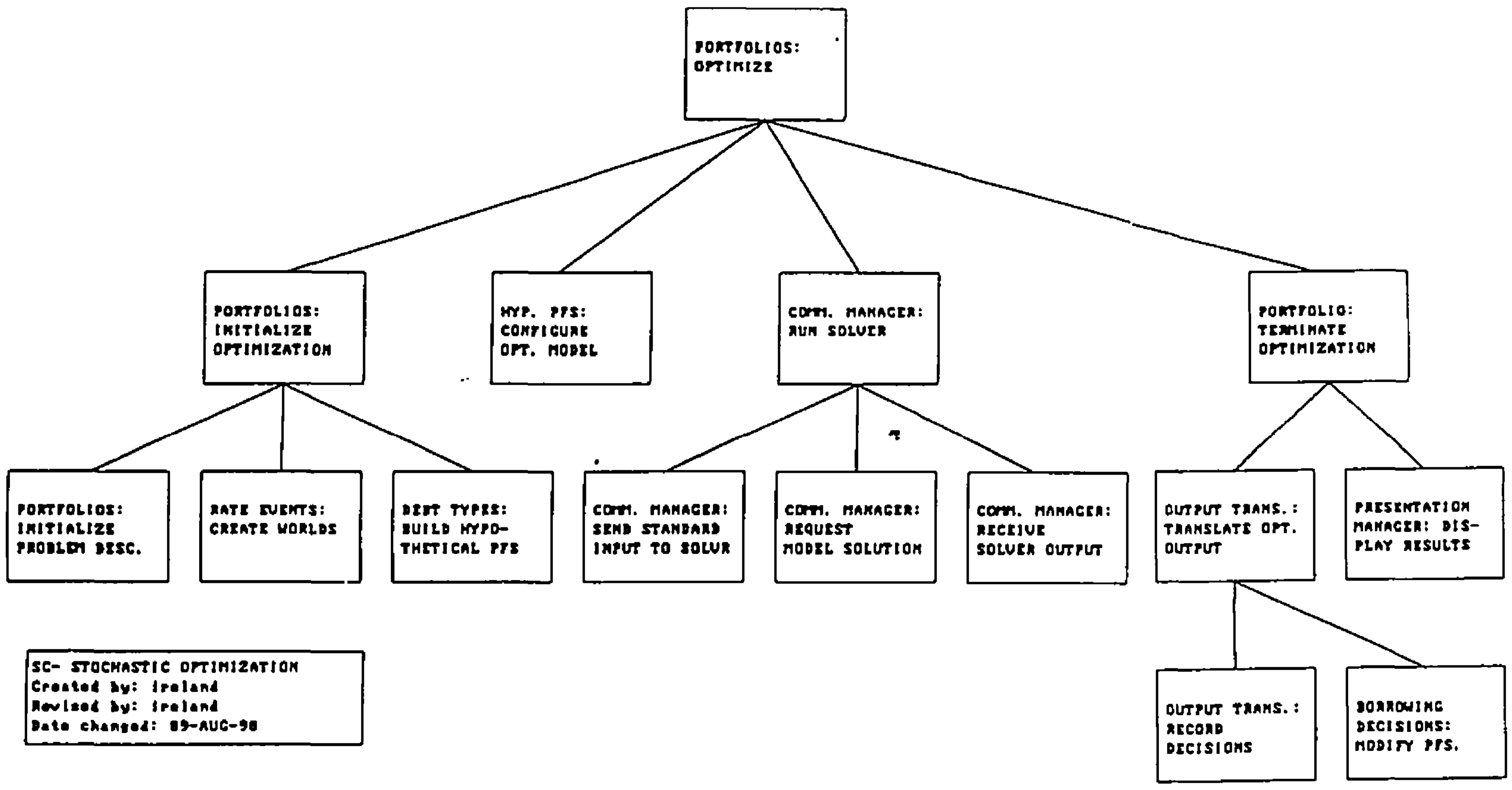

PORTFOLIOS: OPTIMIZE
PORTFOLIOS: INITIALIZE OPTIMIZATION
HYP. PFS: CONFIGURE OPT. MODEL
COMM. MANAGER: RUN SOLVER
PORTFOLIO: TERMINATE OPTIMIZATION
PORTFOLIOS: INITIALIZE PROBLEM DESC.
RATE EVENTS: CREATE WORLDS
DEBT TYPES: BUILD HYPO-THETICAL PFS
COMM. MANAGER: SEND STANDARD INPUT TO SOLVR
COMM. MANAGER: REQUEST MODEL SOLUTION
COMM. MANAGER: RECEIVE SOLVER OUTPUT
OUTPUT TRANS.: TRANSLATE OPT. OUTPUT
PRESENTATION MANAGER: DIS-PLAY RESULTS
OUTPUT TRANS.: RECORD DECISIONS
BORROWING DECISIONS: MODIFY PFS.
SC- STOCHASTIC OPTIMIZATION
Created by: ireland
Revised by: ireland
Date changed: 09-AUG-90

Figure 6.20. Stochastic Optimization Control

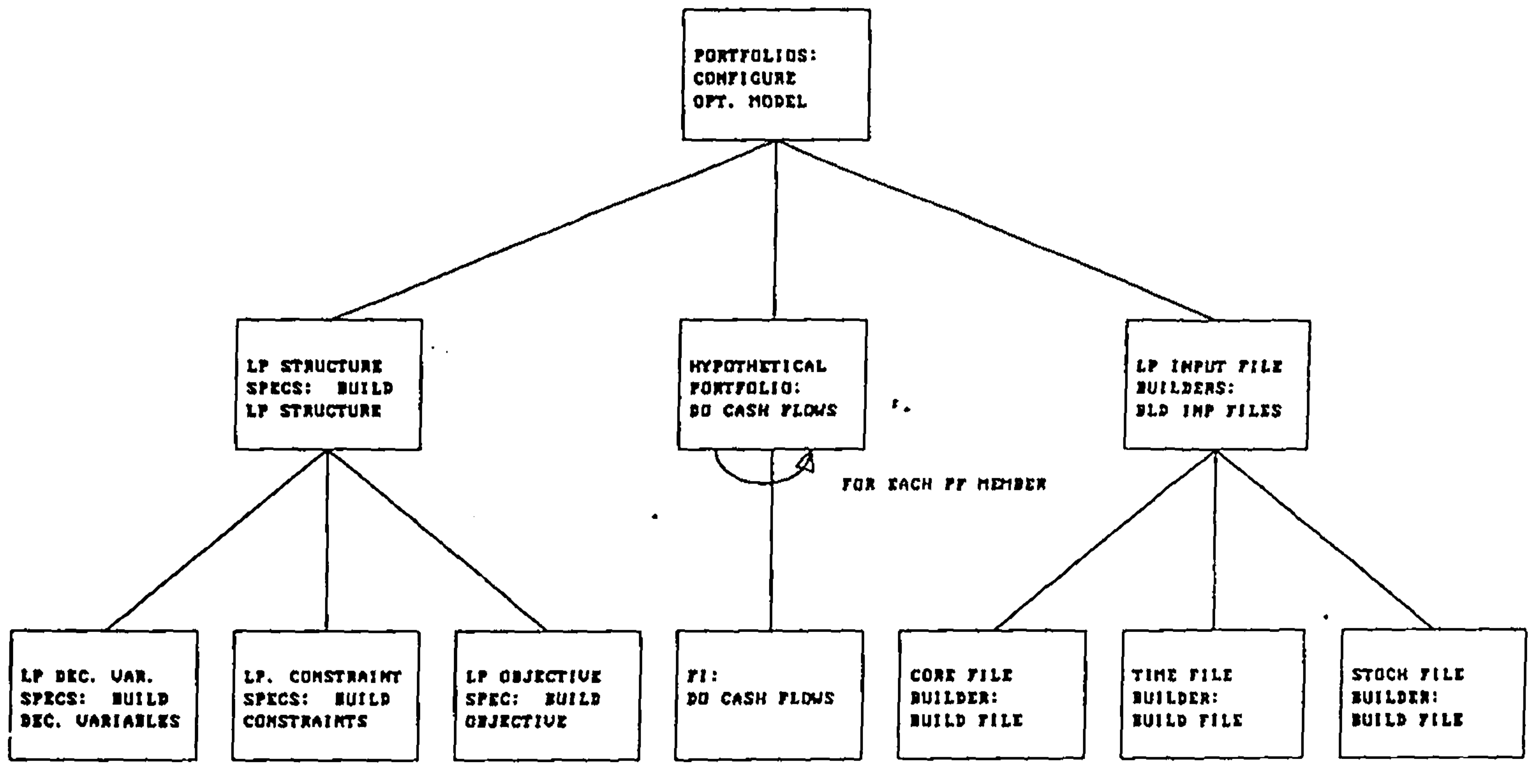

PORTFOLIOS: CONFIGURE OPT. MODEL
LP STRUCTURE SPECS: BUILD LP STRUCTURE
HYPOTHETICAL PORTFOLIO: DO CASH FLOWS
LP INPUT FILE BUILDERS: BLD INP FILES
FOR EACH PF MEMBER
LP DEC. VAR. SPECS: BUILD DEC. VARIABLES
LP. CONSTRAINT SPECS: BUILD CONSTRAINTS
LP OBJECTIVE SPEC: BUILD OBJECTIVE
FI: DO CASH FLOWS
CORE FILE BUILDER: BUILD FILE
TIME FILE BUILDER: BUILD FILE
STOCK FILE BUILDER: BUILD FILE

Figure 6.21. Optimization Model Configuration Control

2. Model configuration

a. Based on general model specifications in LP support objects together with knowledge of the characteristics of the individual debts in the hypothetical portfolio, a hypothetical borrowing action is created for each decision variable. Constraints and the objective function (matrix rows) are then explicitly named according to the MIDAS naming convention (Gassmann and Ireland 1990) and listed in the knowledge base for use by input file builders. At this stage, knowledge of debt characteristics and market conditions is used to eliminate unnecessary decision variables and constraints.

b. In each world, each debt in the hypothetical portfolio calculates its own (per-dollar) optimization coefficients using the methods for rate-list generation and cash flow projection which are also used to produce simulation results.

c. Using the names and specifications for decision variables, the objective and constraints in LP support objects, the debt object coefficients are reformatted to build the required standard input files for the optimization solver.

3. Model solution

a. The portfolio control method directs the communications manager to send the input files to the solver and request solution for the model by running a batch file in the solver machine.

b. This file is executed to solve the model and return the solver output file to the knowledge base.

4. Termination

a. The output file is read, decision variable results are stored as borrowing action amount values in the appropriate worlds, unused debts in each scenario are deleted from their portfolios and debt principal amounts are adjusted to reflect the output values of the optimization decision variables.

b. The complete borrowing plan resulting from the optimization step is displayed, under the control of a presentation manager (see Chapter 7).

For optimization model modifications and reruns, model changes such as constraint revisions are stored and used to alter model input as required. Steps 3 and 4 are then repeated.

3.2 Stochastic simulation control

Figure 6.22 outlines the control structure for stochastic portfolio simulation. Each financial instrument simulates its own financial performance, given future interest and exchange rates; the results of individual borrowing alternatives are modelled using individual financial instrument objects. Portfolios simulate entire borrowing plans by modelling individual financial instruments and combining their results. Simulation model control is handled through methods in financial instruments and portfolios. Detailed simulation output is stored in output tables (see Chapter 7) for presentation and further analysis.

Processing for the simulation model proceeds as follows:

1. Initialization

If the simulation follows optimization and heuristic plan refinement, this step is not necessary. If the simulation is being carried out separately, then model objects, model support objects and worlds are created from user input to fully describe the current problem.

The user may specify the portfolio to be tested by either entering a borrowing plan, which will create a portfolio and its debts, or by directly creating existing and future debt objects and the portfolio.

2. Simulation passes

a. For each market, a sequence of yield curves is randomly generated, as described in Chapter 5, for the planning period from distributions consistent with the event

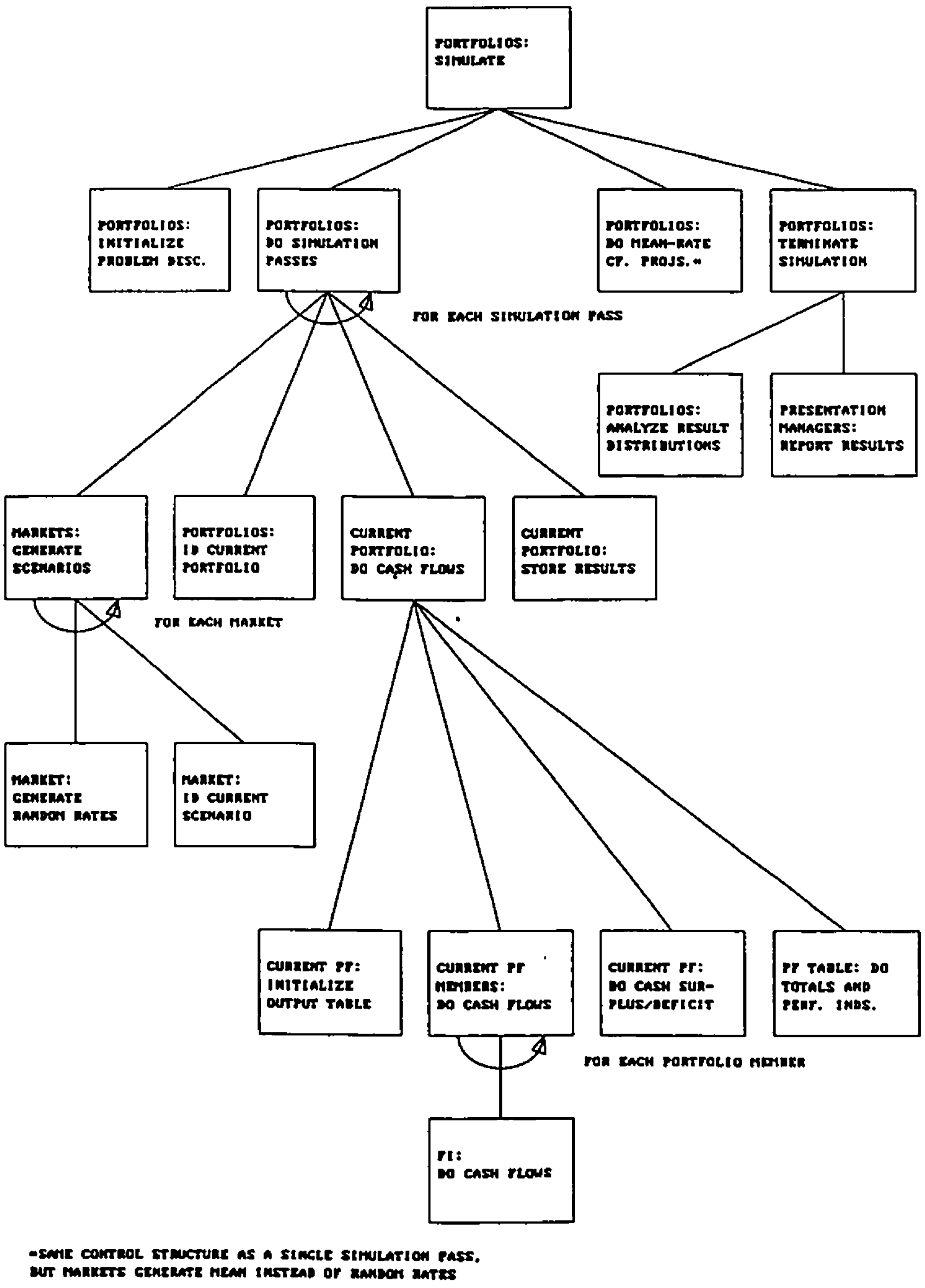

PORTFOLIOS: SIMULATE
PORTFOLIOS: INITIALIZE PROBLEM DESC.
PORTFOLIOS: DO SIMULATION PASSES
PORTFOLIOS: DO MEAN-RATE CF. PROJS.*
PORTFOLIOS: TERMINATE SIMULATION
FOR EACH SIMULATION PASS
PORTFOLIOS: ANALYZE RESULT DISTRIBUTIONS
PRESENTATION MANAGERS: REPORT RESULTS
MARKETS: GENERATE SCENARIOS
PORTFOLIOS: ID CURRENT PORTFOLIO
CURRENT PORTFOLIO: DO CASH FLOWS
CURRENT PORTFOLIO: STORE RESULTS
FOR EACH MARKET
MARKET: GENERATE RANDOM RATES
MARKET: ID CURRENT SCENARIO
CURRENT PF: INITIALIZE OUTPUT TABLE
CURRENT PF MEMBERS: DO CASH FLOWS
CURRENT PF: DO CASH SUR-PLUS/DEFICIT
PF TABLE: DO TOTALS AND PERF. INDS.
FOR EACH PORTFOLIO MEMBER
FI: DO CASH FLOWS
*SAME CONTROL STRUCTURE AS A SINGLE SIMULATION PASS, BUT MARKETS GENERATE MEAN INSTEAD OF RANDOM RATES

Figure 6.22. Stochastic Simulation Control

sequences assumed in the stochastic programming model. The rate path on the scenario tree that corresponds to the generated rates is then identified; it is called the *current scenario* and corresponds to the world in which the simulation pass will take place. (This step and those which follow are under the control of the 'simulate' method in the PORTFOLIOS class.)

b. The portfolio corresponding to the current scenario (the set of debt objects active under this scenario) is identified and called the *current portfolio.*

c. Cash flows are projected for the current portfolio. To do this, the portfolio's output table is first initialized. Cash flows are then calculated according to the defining characteristics and features of the debt (Figure 6.23) and totalled for each member of the portfolio. (The given interest rate curves affect cash inflows and outflows by determining (a) the coupon rate of the new debt, (b) sinking fund earnings rates, (c) short-term debt and cash surplus earnings rates, and (d) the market price of each debt at the end of the simulation period, when it is assumed that all debt is retired.) Any cash deficit or surplus with respect to the stated cash requirements is then calculated and set up as *additional* borrowing or investment. (High-cost penalty borrowing is used to fund deficits.) Portfolio totals and performance indicators, including the ending portfolio value, are then computed.

d. The performance indicator values for the portfolio for the simulation pass are stored as a single observation in the list of simulation results.

The process is repeated as many times as desired to obtain the full performance indicator distributions. Following all passes, cash flow projections are done using mean rates for the entire rate tree to provide details on mean-rate performance for analysis. Finally, results are presented to the user as distribution statistics and graphs, with detailed mean-rate projection results available on request.

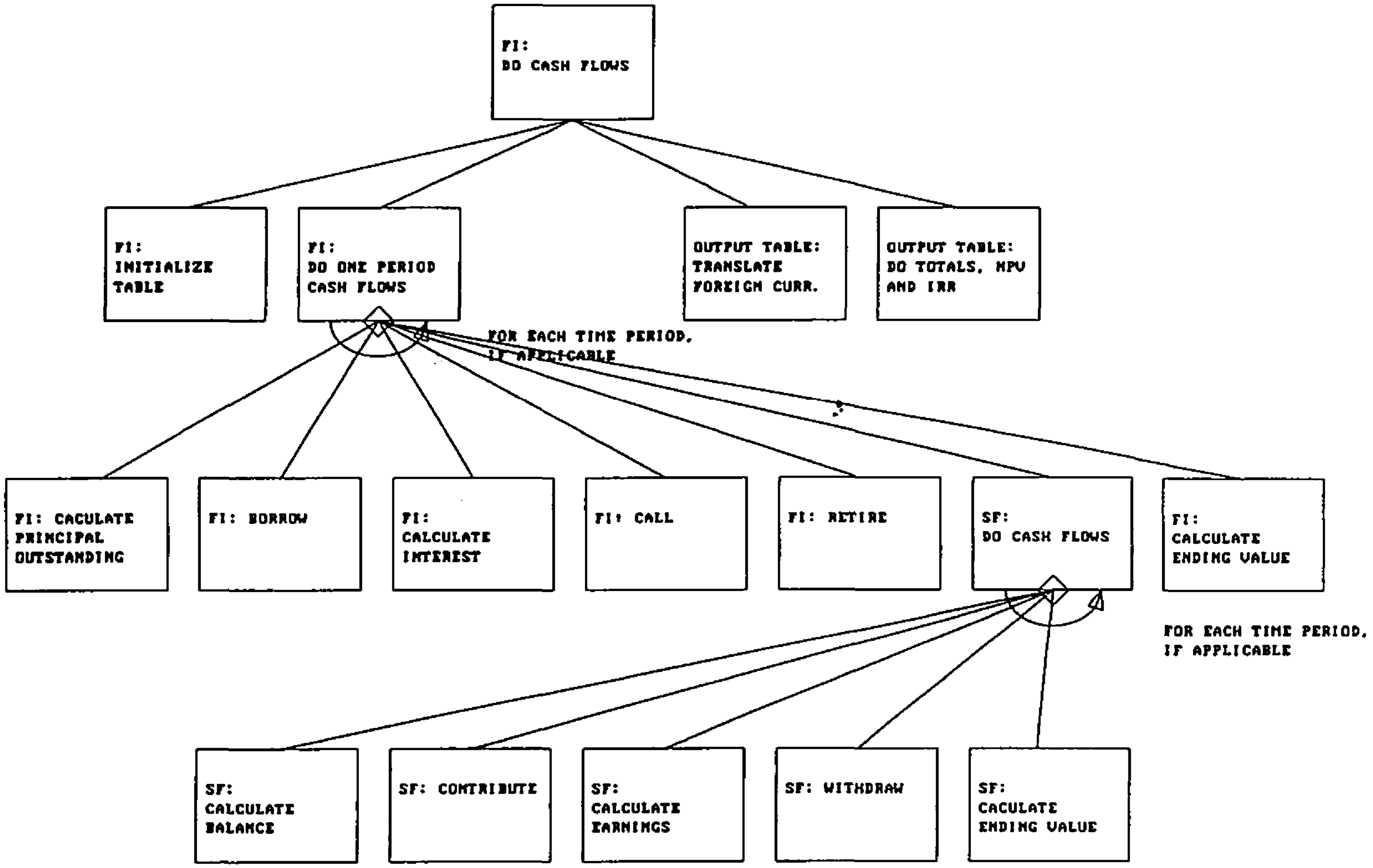


Figure 6.23. Single-Debt Cash Flow Projection Control

3.3 Plan refinement reasoning

Because plan refinement is a data-driven rather than goal-driven process of generating all implications from a given plan and rule set, heuristic rule refinement uses *forward chaining* to inspect and modify optimization output as represented in output tables. Forward chaining begins with the assertion that a given rule set is to be applied to a borrowing plan; the rule or rules in the set are then applied as many times as necessary until all possible conclusions (plan modifications) are made.

4. Implementation status

As indicated in Chapter 4, the implemented system prototype handles heuristic plan refinement and simulation modelling with system support as outlined in Chapter 7. The prototype includes limited debt types, single-path rather than branching rate scenarios and no ability for user creation of new debt types. The prototype is able to carry out full simulation and projection of financial instruments and portfolios, but the model objects (including future debts) must be created directly; debts and portfolios cannot be automatically created by entering borrowing actions. With regard to optimization modelling, coefficients can be generated and the communications interface linking the knowledge base with the optimization solver has been implemented; work is now progressing on LP input file construction and output interpretation.

5. Model performance

Systems implemented in KEE operate entirely in memory (real and virtual); hence their processing speed depends on amount of memory and page space. Optimization performance also depends on the external FORTRAN solver and Ethernet communication if the Explorer rather than the MicroExplorer is used.

Solver performance is discussed in Chapter 5, and optimization formulation has not yet been implemented. Simulation times for 8-debt portfolios with 50 passes are in the range of 15 minutes; simulations of 50 debts, approximating the Corporation's present portfolio, run in approximately one hour.

6. Contributions, strengths and weaknesses of this design

6.1 Research contributions

Both the simulation and optimization models used in MIDAS are highly complex in terms of the number of parameters, algorithms and output values involved and their variations among the numerous financial instruments modelled. Additional complexity arises from the treatment of uncertainty through branching scenarios and stochastic simulation.

As noted in Chapter 2, flexible formulation of complex models such as these has not been demonstrated in decision support systems to date. In addition, object-oriented design and programming have not been applied to complex financial simulation or optimization modelling. This design is therefore innovative for its ability to handle complexity, its use of frame-based, object-oriented techniques in modelling and its ability to achieve formulation of the required optimization and simulation models in a straightforward manner based on a clearly defined domain representation.

Similarly, DSSs have not previously been developed which integrate two major types of complex models as well as rule-based components. As we have seen, MIDAS' use of a common domain representation gives common assumptions, parameter values and low-level calculations such as coefficients for all models, extending integration to include access to common algorithms as well as common data values.

6.2 Overall functionality

As described in this chapter, MIDAS' modelling subsystem is designed to perform all functions outlined at the start of this chapter by maintaining and manipulating its common object-oriented domain/problem representation. Model *formulation* is automatic once the appropriate model objects are created and described; model *solution* is controlled by methods within objects; and model *results alter the domain representation* to maintain the common model structure and current problem description, thereby integrating models and heuristic components. Although the modelling components are specifically crafted for a single domain and problem type, within these constraints the system is flexible in its handling of numerous types of financial instruments, deterministic or stochastic rate forecasts, mean-rate or randomized rate assumptions based on these forecasts, and single- or multiple-debt modelling.

This design makes model and rule integration especially simple and straightforward in MIDAS, despite the complexity of the models involved. The use of domain objects as common underlying conceptual and operational submodels ensures model *consistency* and *communication* in several ways:

1. Both types of models are configured or reconfigured simultaneously when a portfolio is created.

2. The different models have common parameter values for coefficient and simulation/projection calculations, arising from their definition and storage in common model and support objects.

3. The models use the same methods for common calculations even though results are used for different purposes in the modelling process.

4. All model communication, including communication with rule-based reasoning modules, is based on facts in the common objects. Input to and results from each model or rule set are stored as attribute values available to the entire system.

5. Because underlying objects store intermediate domain states, the optimization, refinement and simulation subprocesses are linked to carry out the staged planning process supported by the system. Each has a well-defined, complementary function to perform in adding to or modifying the knowledge base domain description.

Overall, the four model modularization design problems identified in Chapter 5 (module definition, selection, communication and control) are solved in a straightforward manner by the choice of domain objects as basic submodels and by the use of object-oriented programming to implement the models. Definition and selection mimic the creation of an actual debt plan from possible debts; communication is for the most part limited to the passing of data items that would be obtained from the institutions supporting the debt planning process; and control is done through generalized methods which do not have to consider the calculation specifics within objects.

6.3 Other advantages

A domain-specific system organization provides at least two other significant advantages from the developer's and user's viewpoints. First, it provides a clear semantic starting point for communication between system users and developers, since the objects, terms and methods used in the system clearly correspond to those used in the system's real-world domain. Second, the well-documented advantages of object-oriented programming allow the developer to easily build, test and reuse debt submodel definitions and to combine debt objects into portfolio models without specific consideration of modelling details within each portfolio member.

6.4 Design limitations

As now designed, the system's main modelling limitation from a user's viewpoint is its inability to formulate new classes of debt, incorporating innovative interest features or other provisions, under user control. This capability could be provided by redefining all debts or new types as collections of single-feature objects such as basic bonds, call features and sinking fund features, combined under user direction (see Dhar and Pople 1987) for a similar effort for manufacturing simulation); however, this was *not* added to the initial system design because it created extensive additional overhead for linking debt components into coherent models.

The system's main weaknesses from the developer's perspective are those of object-oriented programming and LISP: poor performance, the need for extensive computing resources and difficulties in managing the knowledge base. These are discussed in more detail in Chapter 9. In addition, the many methods which implement the modelling algorithms illustrated in this chapter are highly interdependent as to their order of execution and dependent for correct results on the knowledge base conditions required at any point during modelling, so that method maintenance is relatively difficult. (Inheritance does, of course, reduce the need for method changes.) Further work on ways to improve error-checking and reduce this interdependence would improve the maintainability of the model.

7
System Support Design

MIDAS' system support subsystem handles the functions necessary for the system to be able to interact with its users and to control its operations. Significant portions of this subsystem illustrate a data-driven control approach which adds great power and flexibility to functions which would otherwise be quite cumbersome and resource-intensive to implement. This chapter describes the three functional components comprising the system support subsystem: the ***user interface***, including windowing, screen layout and user request processing; ***output management***; and overall ***system control*** including ***task scheduling***. Figure 7.1 presents an object diagram for the entire system support subsystem; each component is discussed separately below.

1. The user interface

1.1 Functional requirements

The system's user interaction utilizes the mouse-menu interface provided by KEE and the Explorer, expanded to include the form-based input style expected by business users such as Corporation management and staff. The user interface is designed to provide the following functional support for user-system interaction:

1. Production of display screens, including a constant background and various prompt and output windows.
2. Menu-based handling of user requests through icons, menus and mouse-activated method or rule calls.
3. Form-based object maintenance and problem description input.

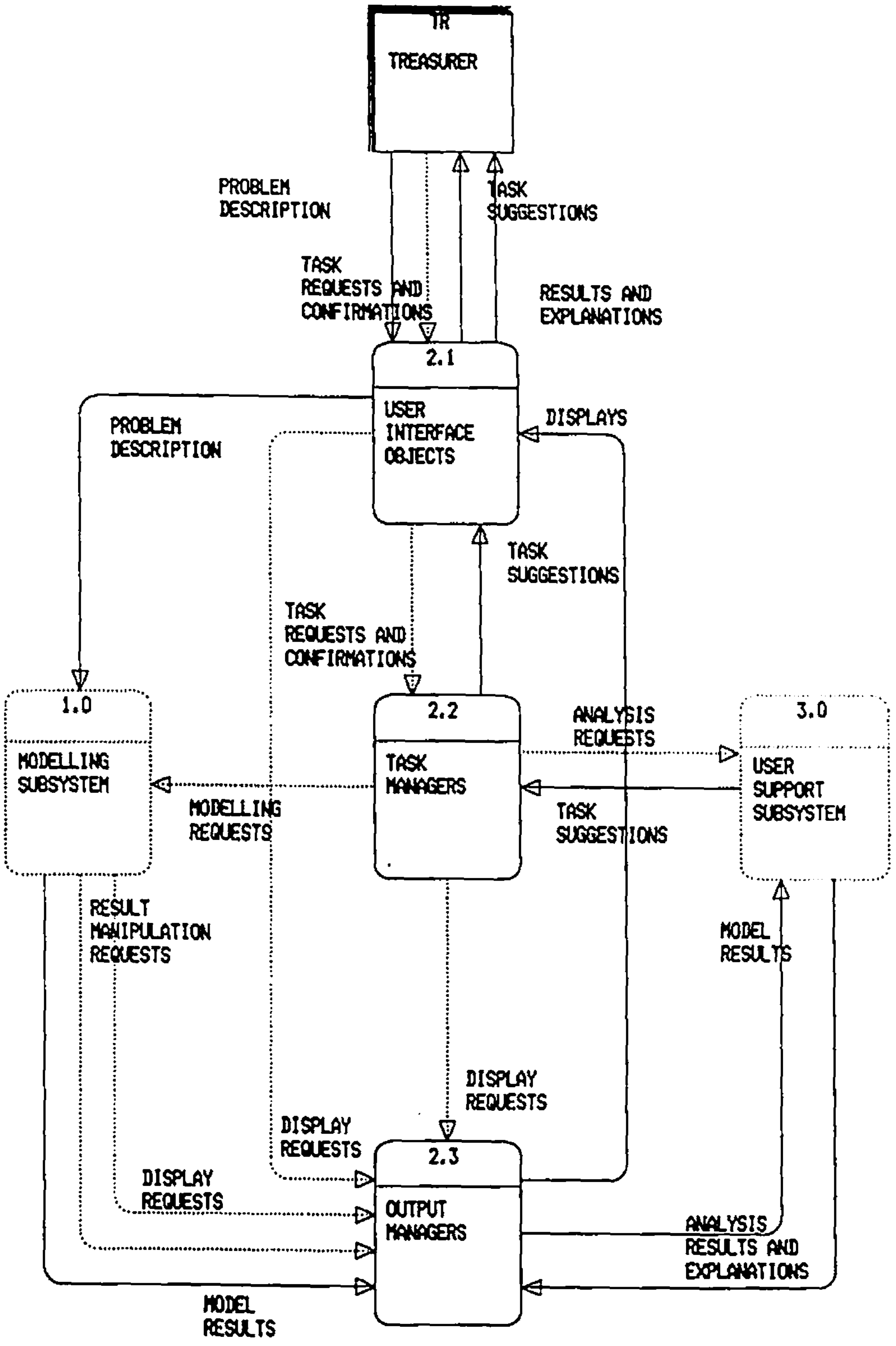


Figure 7.1. System Support Subsystem

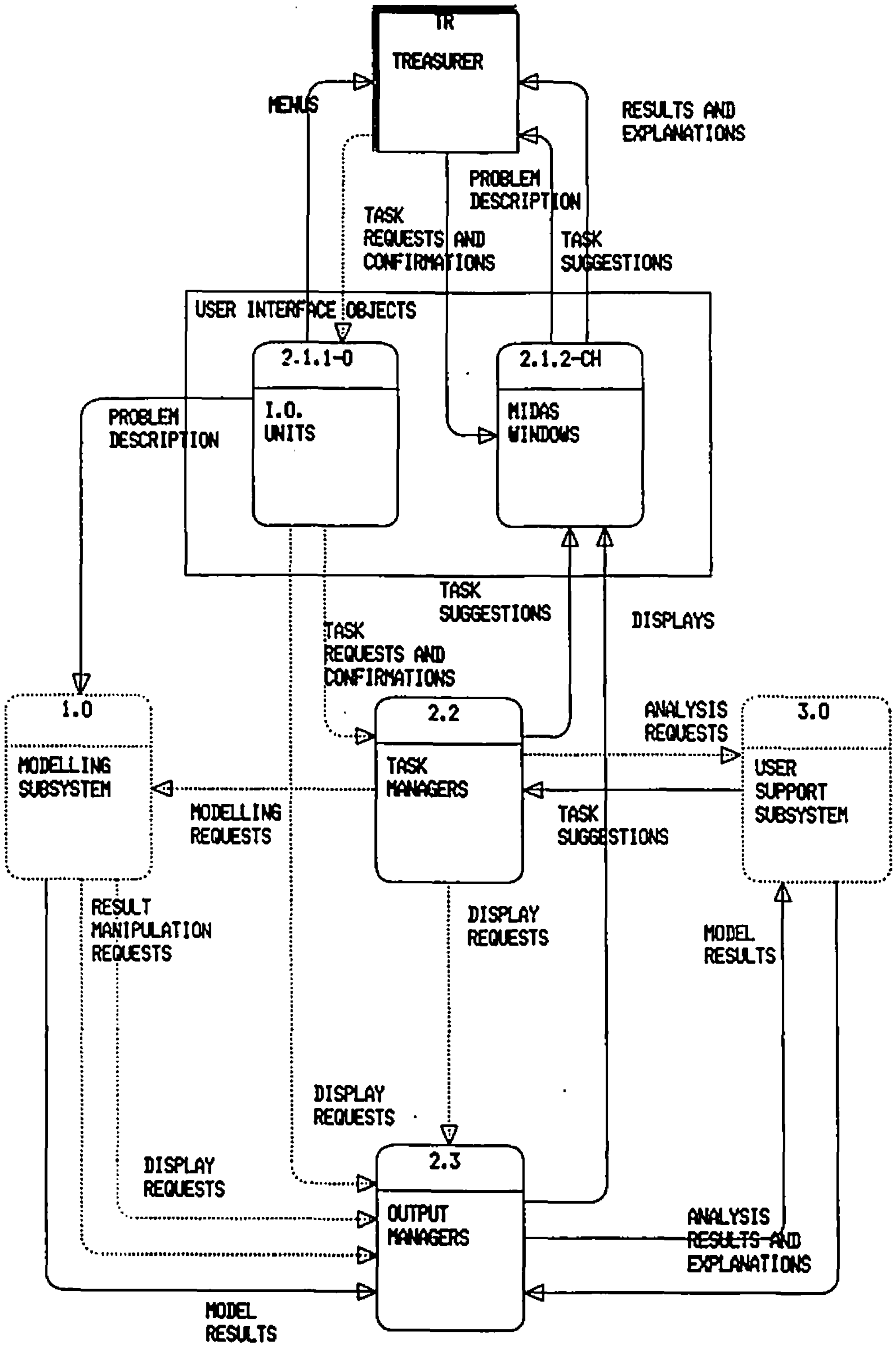


Figure 7.2. User Interface Object Diagram

Because these items are provided by KEE and Explorer software functions, they are described only briefly with examples below.

1.2 Display screen, window, icon and menu production

Figure 7.2 shows user interface objects in the context of the system support subsystem. KEE and the Explorer operating system provide LISP functions for screen, window and icon generation. Screens, windows and icons (represented generically in Figure 7.2 as 'MIDAS windows' are defined as KEE objects using the KEEPictures facility (Intellicorp 1988*c*), Common Windows (Intellicorp 1988*b*) and Explorer windows functions (Texas Instruments 1987). Mouse-click responses are specified when icons are created, and screen and window displays are handled by windows functions in methods in the *I.O.unit*, a single object which organizes menu and window-related slots and methods. Menus are produced from specification lists in slots in the I.O.unit, which enumerate the method calls to be used in response to mouse activated menu selections.

The system's main display screen is illustrated in Figure 7.3. Interaction is begun by clicking on the MIDAS icon in the top left corner. This produces a *cascading menu* in which succeeding levels of functions appear when the mouse is dragged through menu lines; another mouse click selects the appropriate function choice.

1.3 Form-based input

Problem description input, which results in the creation of object instances and the filling in of their slot values, is done through *choose-variable value* forms defined in the Explorer window system (for an example see Figure 7.4). The forms are controlled by LISP *add*, *change* and *delete* methods in each object class; with these methods, each class of model object is able to create instances of its class, fill in slot values, modify instances and delete them.

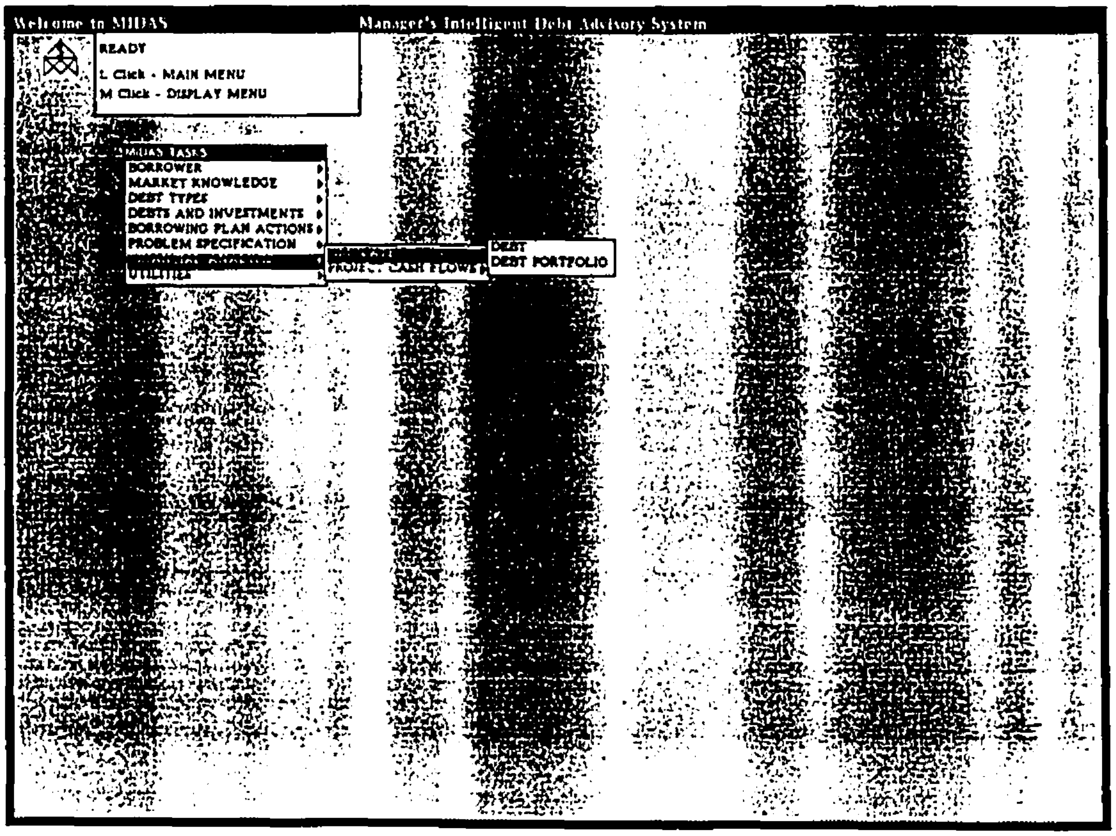
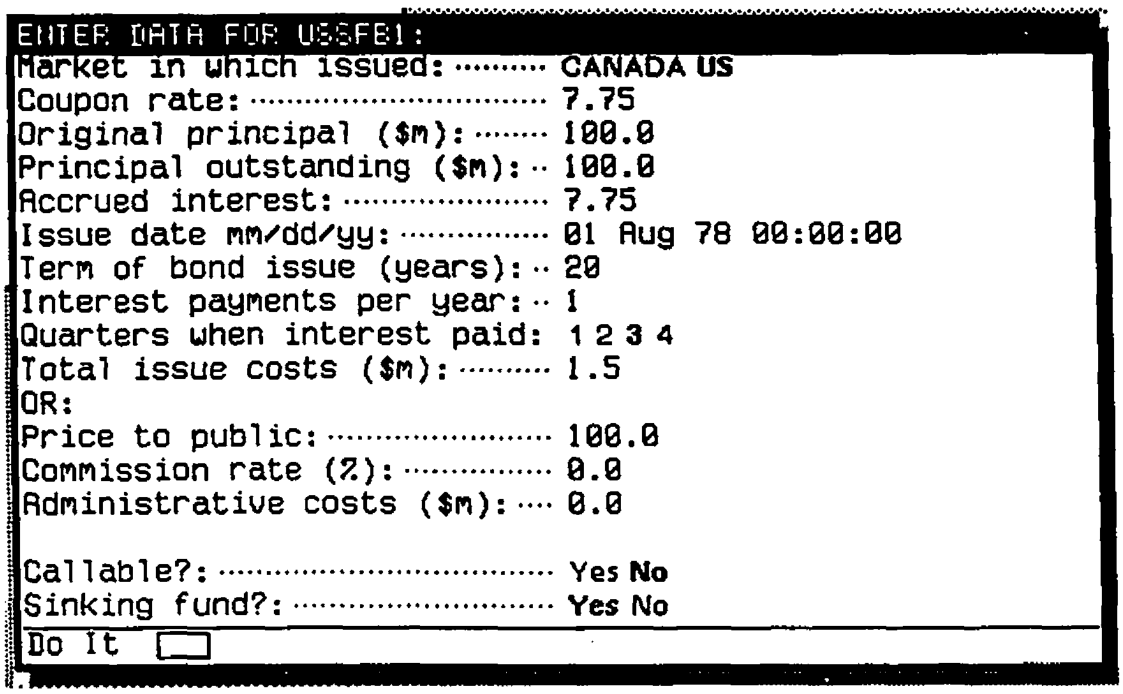


Figure 7.3. MIDAS Main Display Screen Showing Menu Options



```
ENTER DATA FOR USSFB1:
Market in which issued: ·········· CANADA **US**
Coupon rate: ······························ 7.75
Original principal ($m): ········ 100.0
Principal outstanding ($m): ·· 100.0
Accrued interest: ······················ 7.75
Issue date mm/dd/yy: ················ 01 Aug 78 00:00:00
Term of bond issue (years): ·· 20
Interest payments per year: ·· 1
Quarters when interest paid: 1 2 3 4
Total issue costs ($m): ·········· 1.5
OR:
Price to public: ······················· 100.0
Commission rate (%): ················ 0.0
Administrative costs ($m): ···· 0.0

Callable?: ·································· Yes **No**
Sinking fund?: ··························· **Yes** No
Do It
```

Figure 7.4. Choose-Variable-Values Input Form

1.4 System housekeeping

Necessary housekeeping tasks such as saving current problem states and moving between the user interface and the development system are requested through a *utilities* menu item and are handled by KEE or LISP functions in various objects as appropriate.

2. Output management

Both the optimization and simulation models used in MIDAS produce large quantities of numerical output which must be organized, stored, manipulated and presented efficiently. The Explorer and KEE do not provide built-in functions for these output management functions, so the MIDAS design includes specifications for object-oriented output management.

2.1 Functional and knowledge requirements

MIDAS' extensive model results form a primarily numeric database which requires extensive manipulation in the course of modelling, analyzing results and presenting them to the user. Specific output management functions include:

1. Representation and organization of multi-line time series data such as cash flow details over any given planning period.
2. Commonly-used numeric manipulation and calculation operations (initialization, totalling, foreign exchange translation, net present value, internal rate of return and statistical analysis).
3. Cross-referencing of result data to and from its sources (financial instruments and portfolios).
4. Table, graphic and text-oriented result presentation.

Knowledge and processing requirements for these functions are relatively simple; with the exception of the specific display and reporting functions for the Explorer system, all are routinely handled in business data processing and financial modelling systems. However, the diversity and large volumes of data to be handled in MIDAS make efficient, easily-maintainable output handling important to the overall system design.

2.2 Knowledge representation and control approach

Because they are similar across large and diverse groups of data, these functions can be managed most efficiently using an approach similar to that of database management systems which manage data based on descriptive *metadata* (data about the data in the object) and *generalized operations*. Such an approach guides MIDAS' output management component, shown in Figure 7.5 in the context of the system support subsystem. Within this component, all model output data is stored and manipulated in generic *output table* objects; *presentation managers* use data in the output tables to produce specific output displays, reports and graphs.

2.3 Output tables

Output tables store time-dependent cash flow projection output, used for optimization coefficients and simulation results, by line item in LISP lists. (Because KEE does not explicitly support arrays, the lists are used to mimic arrays with each indexed position corresponding to a time period (quarter or year) between period 1 and the end of the planning horizon.)

Output tables are defined in classes for single financial instruments and for portfolios and are instantiated for each portfolio and basic model object in a problem; each foreign debt has separate tables for foreign and domestic-currency results. Financial instrument tables hold and manipulate results for an individual basic model object, while portfolio tables hold and manipulate line-by-line portfolio result totals across objects in the portfolio as well as performance indicators for the portfolio as a whole. In addition to holding result lists, output tables contain generic methods

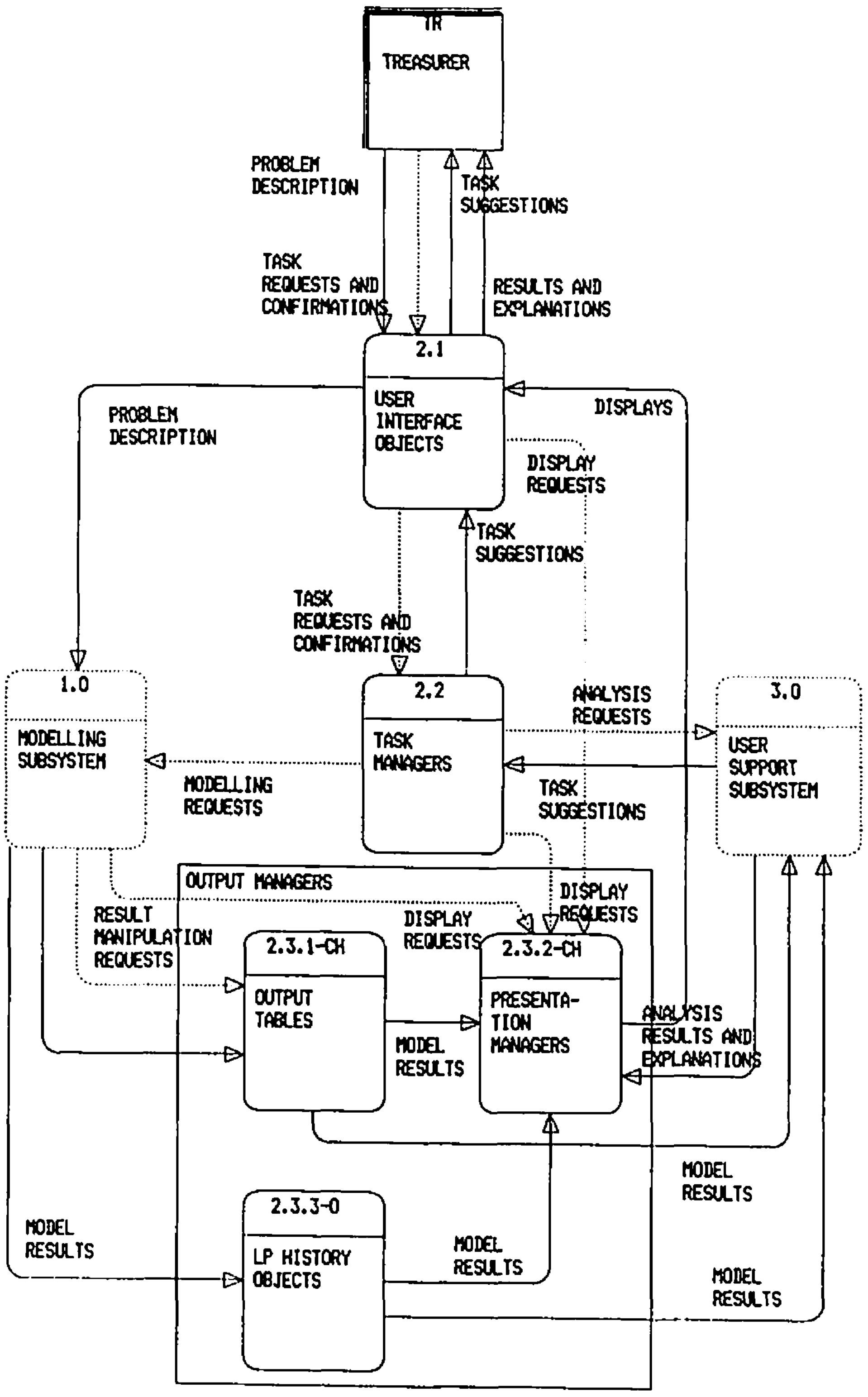


Figure 7.5. Output Management Object Diagram

and specifications for manipulating list contents in commonly required ways such as summation, statistical analysis and foreign exchange translation.

Output tables are large objects which contain a number of types of slots which have specific output manipulation roles. Each slot type has associated facets (slot attributes) which drive general output manipulation methods for that slot type. Slot types include the following:

1. *Cash flow slots.* For basic model objects, these slots hold lists of cash flow projection results by time period produced by calculation methods within the model objects. For portfolios, cash flow slots hold totals by time period for slots of the same name across all members of the portfolio. Examples of cash flow slots include principal borrowed, principal retired, principal outstanding, interest paid, sinking fund balance and sinking fund contributions. Each cash flow slot has an *active* facet which contains a flag indicating whether the slot is used by the particular model object which generates results for that table. Figure 7.6 gives an example of a cash flow slot in an output table.

Slot name	cf.principal.outstanding
Value	(0 250 250 275 225 200)
Facets:	
active	T
initial.element	0.0
title	"Principal Outstanding"

Figure 7.6. A Cash Flow Slot in an Output Table

2. *Memo slots.* These slots hold time-dependent non-cash flow data which is either needed during cash flow projection or helps to explain projection results. Examples are a status slot which indicates whether a debt is issued, active, callable or mature in each time period; and lists of market interest rates and foreign exchange rates applicable to a particular cash flow projection.

3. *Total slots.* Total slot values in an output table are sums by time period of values in other slots in the table. Each total slot has a *sum.of* facet containing names of the slots to be summed by an associated general method to give the slot value. Total cash flows and total operating cash flows are examples of total slots. Figure 7.7 shows a total slot.

Slot name	cf.total.cash.flows
Value	(500 550 700 -500 -500 -1000)
Facets:	
title	"Total Cash Flows"
active	T
initial.element	0.0
sum.of	(cf.principal.borrowed, cf.issue.costs, cf.interest.paid, cf.retirement.costs, cf.principal.retired, sf.contributions, sf.withdrawals)

Figure 7.7. A Total Slot in an Output Table

4. *FX slots.* These are slots whose values are to be translated from foreign currency to Canadian dollar values.

5. *NPV slots* contain the net present values of lists elsewhere in a table. Each NPV slot contains a *base slot* facet which names the slot to be discounted by the general NPV method.

6. *IRR slots* contain the internal rates of return of lists elsewhere in a table. Each IRR slot contains a *base slot* facet which names the slot to be analyzed by the general IRR method. The IRR is calculated iteratively and a value of NIL placed in the slot if the IRR is multiple-valued.

7. *Performance indicator slots.* These slots, found only in portfolio tables, hold lists by time period of portfolio performance indicators such as average debt term, percent of long-term debt, percent of foreign-currency debt, and percent of

variable-rate debt. Performance indicators are averages, percentages or ratios of selected values elsewhere in the table; the general performance indicator method is driven by the following facets:

a. The *numerator* facet, which names a slot to be accumulated across portfolio members for the numerator of the performance indicator.
b. The *denominator* facet, which names a slot to be accumulated across portfolio members for the denominator of the performance indicator.
c. The *multiplier* facet, which names a slot whose values multiply numerator values, for performance indicators that are weighted averages.
d. The *percent* facet, which has a value of T if the indicator is a percentage (requiring multiplication by 100) and NIL if the indicator is not a percentage.
e. The *condition* facet, which contains conditions (in the form of LISP expressions to be evaluted) which must be met for a portfolio member to be included in the numerator of the performance indicator.

Figure 7.8 gives an example of a performance indicator slot and outlines the performance indicator calculation method.

8. *Constraint warning slots*, which contain lists of differences between constraint values and list values produced by the simulation. These are calculated by a general method which gives a list of differences between a *base.slot* value and a *constraint* value in the knowledge base.

9. *Simulation result slots.* These slots contain lists of simulation results, such as ending portfolio value, for multiple simulation passes. Each simulation result slot also has facets for holding statistics (*maximum, minimum, mean, standard deviation* and *quartiles*) calculated from the slot's values by a general result analysis method. Figure 7.9 illustrates a simulation result slot.

10. *Name list slots.* Each of these slots contains a list of names of slots in one of the above categories. For each slot category, the list is used as an argument to

Slot name	cf.percent.st
Value	33.0
Facets:	
title	"% Short-Term Debt"
initial.element	0.0
numerator.slot	cf.principal.outstanding
multiplier.slot	nil
denominator.slot	cf.principal.outstanding
condition	($\leq$ term 1.0)
percent	T

Method name:	mth.perf.indicators

Method value:

```
FOR EACH SPECIFIED PERFORMANCE INDICATOR
    GET FACET VALUES FOR INDICATOR SLOT
    IF MULTIPLIER.SLOT NIL
    SET MULTIPLIER.SLOT VALUE TO 1.0
    FOR EACH MEMBER OF PORTFOLIO
        IF CONDITION TRUE FOR MEMBER
           GET VALUES FOR NUMERATOR.SLOT,
           DENOMINATOR.SLOT, MULTIPLIER.SLOT (IF NOT NIL)
           FOR EACH TIME PERIOD $T$
                ADD NUMERATOR VALUE TIMES MULTIPLIER
                VALUE TO NUMERATOR TOTAL AT $T$
                ADD DENOMINATOR VALUE TO DENOMINATOR
                TOTAL AT $T$
SET PERFORMANCE INDICATOR EQUAL TO NUMERATOR VALUE /
DENOMINATOR VALUE
IF PERCENT, MULTIPLY PERFORMANCE INDICATOR VALUE BY 100
```

Figure 7.8. A Performance Indicator Slot and Method in an Output Table

Slot name	sim.end.value
Value	(15 15.2 15.5 20 29 22 16 19)
Facets:	
title	"Ending Value"
base.slot	pf.end.value
high	29
low	15
mean	18.96
std.dev	3.5
quartile.1	15.2
quartile.2	16
quartile.3	20

Figure 7.9. A Simulation Result Slot in an Output Table

the associated general output manipulation method resulting in its application to all slots in the list. For example, the *total.slots* value lists all slots in the table for which values are to be calculated using the *totals* method.

In addition to the facets described above, slots in all of the above categories except the last have a *title* facet which contains its title for display and reporting purposes and an *initial.element* facet which contains the symbol (0.0, 0 or NIL) to be used to initialize the list at the start of a modelling operation.

The data manipulation methods provided in output tables do not operate as part of a single task but are called as required by model solution and output analysis methods. During simulation, for example, each table initializes itself on request from the portfolio or debt; initialization involves zeroing all lists and setting facet values (depending on debt type) indicating which lists are active during the current simulation. The tables serve as references for the model objects during the simulation; each simulation pass fills in the tables with cash flow details, and totals are calculated by the tables when the details have been calculated. Foreign exchange translation is then done to convert each foreign-currency table to a Canadian-dollar set of results. Portfolio totals are accumulated by the portfolio table based on the currently active cash flow slots in the portfolio. The table also calculates NPVs, IRRs and performance indicators at the end of each pass and statistically analyzes result distributions after all passes are completed.

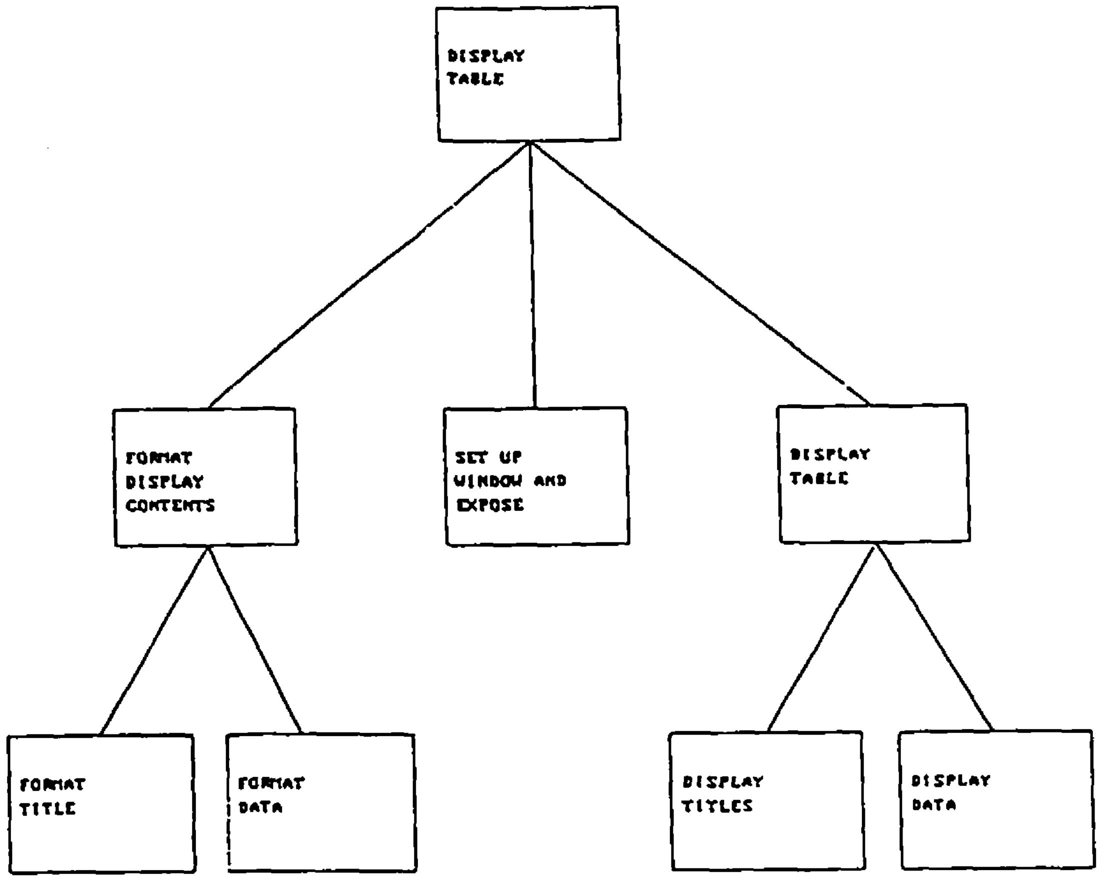
2.4 LP history objects

LP history objects maintain records of user or system-generated modifications and reruns of a model, for use in heuristic plan refinement, EVPI-based model modification, parametric analysis and key factor impact analysis. Each object contains before-and-after records of all objects and slot values changed from the last version of the model, including rate events; the date and time of the resulting run; and the objective value (for optimization) or main performance indicator values (for simulation) resulting from that run. For parametric or key-factor impact analysis, only one object records the results of the sequence of parameter changes and reruns.

2.5 Presentation management

Output presentation is handled by *presentation manager objects*. Presentation managers have their own class hierarchy, which takes advantages of inheritance to minimize code required to produce variations in output displays. Separate classes of presentation managers handle table displays of result lists, displays of attributes for individual model objects, graphs, text output and printed reports, taking data values from output tables or other objects as necessary. Generic presentation methods using Explorer and KEE windowing and graphics functions format and display output; a manager instance defined for each display inherits these methods and contains additional slots specifying display contents and source. Output specifications such as data sources and destination window names are explicitly stored as slot values rather than being included in procedural code.

Output presentation methods follow relatively simple control structures prescribed by KEE and the Explorer windowing system. Table scrolling, necessary to



NOTE: ALL METHODS ARE CARRIED OUT BY TABLE DISPLAY MANAGERS

Figure 7.10. Table Presentation Control Structure

show tables larger than will fit into windows, is handled by window ***repaint*** functions. A table presentation method structure chart is shown in Figure 7.10.

2.6 Contributions, strengths and weaknesses of this design

This general-purpose management scheme for model output demonstrates a straightforward form of object-oriented data management by explicitly representing data management knowledge in objects and manipulating the tables with generic procedures defined by object class. The approach has several advantages. First, it simplifies the structure of model objects by limiting them to definitional, attributes, maintenance methods and unique calculation methods; their results are stored separately in dedicated tables. Second, it significantly reduces the code required to carry out the extensive, generic calculations required of the models. Third, it provides much self-documentation of the system, since operation specifications are clearly detailed in slot and facet values. Finally, it allows straightforward maintenance and extension of many table operations through simple slot value changes rather than procedural code alterations.

The main weakness of this design is wasted space within tables. A single table definition is used for all types of financial instruments and many slots are left empty by some types of debt. This could be corrected by defining new output table subclasses corresponding to individual debt types and containing only the necessary slots for each type.

Another weakness of the representation scheme which may or may not prove significant as the system is expanded is the use of lists within single slots to represent time-dependent data sequences. This representation creates some data redundancy when worlds are implemented, since it requires a complete copy of a list to be stored in a new world if any element in the list varies between that world and its parent world. Alternative representations, including pointers to values within a relational database either within the Explorer or on distributed hardware, will therefore be investigated as part of the implementation of scenarios and worlds in the system.

For report generation, the explicit representation and generic manipulation of knowledge about display types provides significant advantages over explicitly coding individual display procedures. Although the Explorer and KEE do not provide high-level business reporting and graphics capabilities, this approach has enabled the construction of some generic business table and graphics functions. Because new display types are created as specializations of existing types, inheritance minimizes the new coding required. Furthermore, understanding and maintenance are relatively straightforward due to the data-driven and self-documenting nature of the representation.

3. System control and task management

3.1 Functional and knowledge requirements

MIDAS' overall task execution sequence is designed either to be controlled by the user through menu requests or to be system-suggested with user confirmation. In the latter operating mode, the system is intended to mimic a human advisor, leading the user through the complete planning process, suggesting and explaining steps and executing them on user approval.

Specifically, the task management component should accomplish the following:

1. Suggest the next task at any point in a planning session, based on the current stage of problem analysis and whether or not all prerequisites for a task have been met.
2. Explain to the user the purpose of the task and why it is suggested.
3. Ask for user approval or modification of the task sequence.
4. Carry out the specified task.

These functions require the following types of knowledge:

1. Knowledge of required or suggested task execution order.

2. Preconditions for each task.

3. Task descriptions and explanations of purpose.

4. Prompts to ask for user responses.

5. Planning procedures for deciding when to execute tasks.

6. Procedures for carrying out each task.

3.2 Knowledge representation

The task management component is designed as a rule-based agenda management system in which (a) objects explicitly represent tasks and task conditions, and (b) generalized rules dynamically maintain a list of planned tasks and execute current tasks based on the knowledge base state and analysis history. This design is a modification of an approach suggested by Winkelbauer (1988) in which model descriptions and requirements are explicitly represented in objects to support rule-based model management.

Figure 7.11 shows object relationships for the MIDAS task management component within the context of the system support subsystem. Objects in the subsystem include ***task objects***, which contain task requirements, purposes, conditions, prompts and execution procedures; ***task condition objects***, which organize condition descriptions, checking procedures and correction procedures; and a ***task controller***, which holds the agenda, identifies the current task and executes it when requested or confirmed by the user.

3.3 Task objects

A task object example is shown is Figure 7.12. Each task contains names of conditions to verify before a task is executed, a task description (for explanation purposes), two levels of prompts for user confirmation that the task should be executed, subtasks to be checked and added to the agenda if the task is to be executed, a

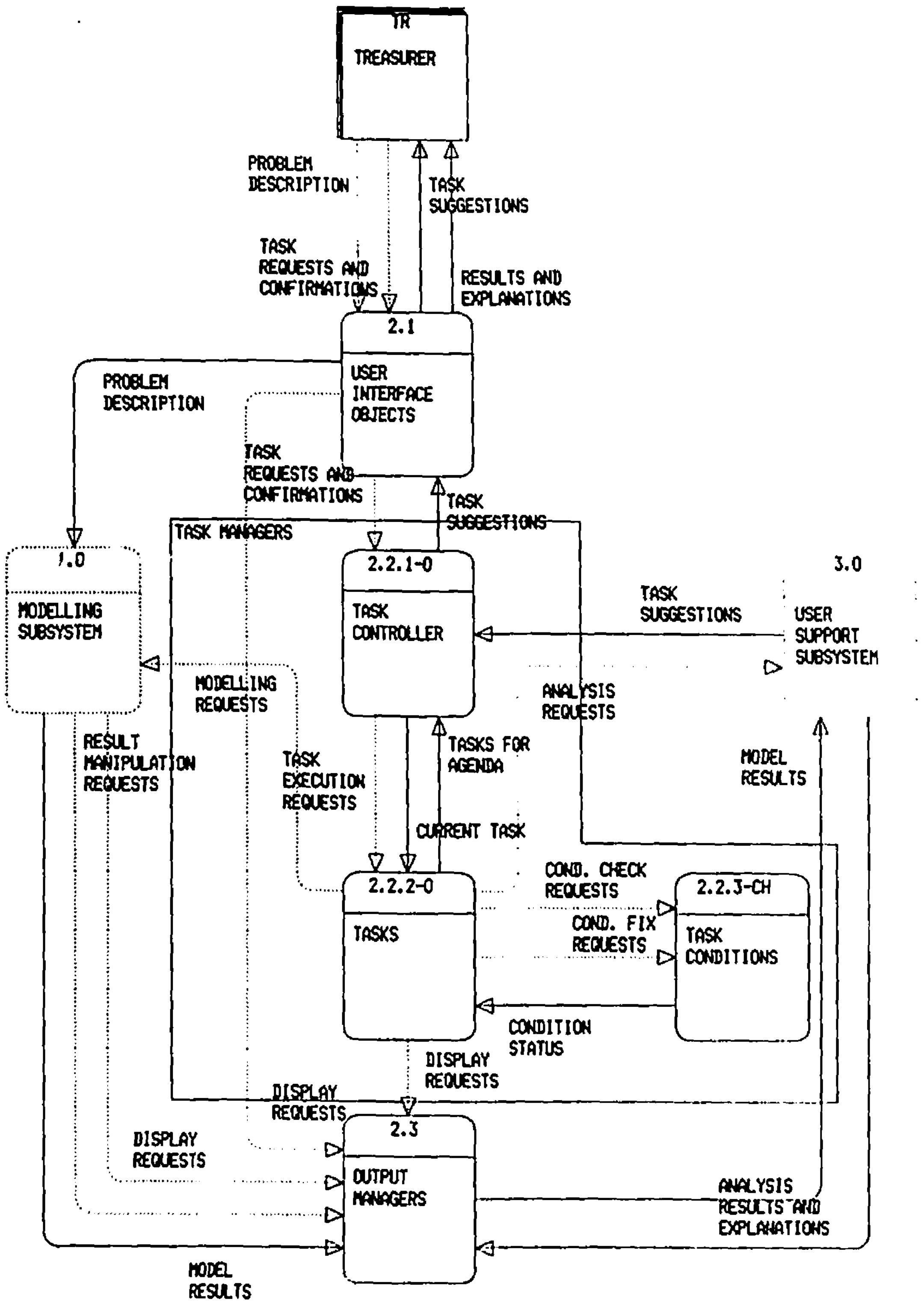


Figure 7.11. Task Management Object Diagram

NAME:	BUILD.LP.INPUT
SUBCLASS.OF	NIL
INSTANCE.OF	TASKS

OWN SLOTS:

alternate.task	SAVE.PROBLEM.SPACE
conditions	(LP.STRUCTURE.EXISTS LP.PF.EXISTS)
subtasks	(BUILD.CORE.FILE BUILD.TIME.FILE BUILD.STOCH.FILE)
next.task	SOLVE.LP
long.prompt	"We now must build input files for the optimization model solver. The process may take several minutes. OK to proceed (Y/N)?"
short.prompt	"Build LP input (Y/N)?"
confirm?	*T*
required?	*T*
schedule.status	CURRENT
condition.status	*T*
confirmed.status	UNKNOWN
performed.status	DONE
subtask.status	UNKNOWN
why.perform?	"The optimization model requires input in three specific files containing (a) coefficients and decision variables for the 'base' rate scenario, (b) data on the time structure of the model and (c) coefficients that vary from the base scenario in other scenarios. This task builds those input files for use by the optimization solver."

METHODS:

check.conditions
confirm
perform.task

Figure 7.12. A Task Object

suggested alternative task to be executed if the task is rejected, the name of the task to be executed following this task, and several status indicators used by the task control rules. It also contains methods for confirming the task with the user and for executing the task by sending an appropriate message.

- 1.0 Initialize.problem
  - 1.1 Load/create.problem.space
  - 1.2 Change.problem.specification
  - 1.3 Set.prompt.type
  - 1.4 Set.analysis.type (full.plan.creation, plan.simulation, pf.simulation, fi.simulation)
- 2.0 Review.background.knowledge
  - 2.1 Review.markets
  - 2.2 Review.debt.types
  - 2.3 Review.existing.debts
  - 2.4 Check.opening.balances
- 3.0 Describe.current.problem
  - 3.1 Enter.borrower.requirements
  - 3.2 Enter.future.rates
  - 3.3 Select.debt.types
  - 3.4 Enter.new.debts
  - 3.5 Enter.borrowing.actions
  - 3.6 Build.portfolio
- 4.0 Optimize.plan
- 5.0 Do.EVPI.analysis
- 6.0 Do.parametric.analysis
- 7.0 Do.heuristic.refinement
- 8.0 Do.simulation.or.projection
- 9.0 Do.key.factor.analysis
- 10.0 Explain.results
- 11.0 Save.problem.space
- 12.0 End.analysis

Figure 7.13. MIDAS Task Tree

3.4 Task controller

The task controller contains the current ***task agenda*** as a list in which the first element is the current task. Suggested task order for the entire system (Figure 7.13) is maintained in a tree structure by the *next task* and *subtasks* slots in task objects. A task is added to the agenda when: (a) it is the next task following an executed task; (b) it is a subtask of a task approved for execution; (c) it is a task required to correct a condition for execution of a task already on the agenda; or (d) it is requested by the user. A task is removed from the agenda when it is finished or rejected by the user. A task is omitted when it is not appropriate for a particular type of analysis.

3.5 Task conditions

A task is only executed if it is on the agenda and satisfies the following: (a) all necessary input is present in the knowledge base; (b) the problem description is internally consistent; (c) no other conditions are present which indicate that it should not be performed; and (d) it is either a required task or is confirmed by the user if it is a task that should be confirmed. Confirmation is handled directly using task prompts; other conditions are explicitly represented in *task condition* objects and checked using generalized methods and rules.

Task conditions are defined in a class hierarchy (Figure 7.14). *Input conditions* check whether categories of input are present, listing relevant objects and confirming them with the user. *Consistency conditions* check relationships among knowledge base parameters for consistency (e.g. the same date for opening debt balances and the start of the modelling period). *Contra conditions* check for miscellaneous conditions indicating that the task should not be done (e.g. model size will be large and user is not willing to wait for the time required). If a condition is not satisfied and cannot be confirmed by the user, a task to correct it is added to the agenda. Condition status remains as set once checked unless reset by specified changes made to the knowledge base. Figure 7.15 shows a task condition object.

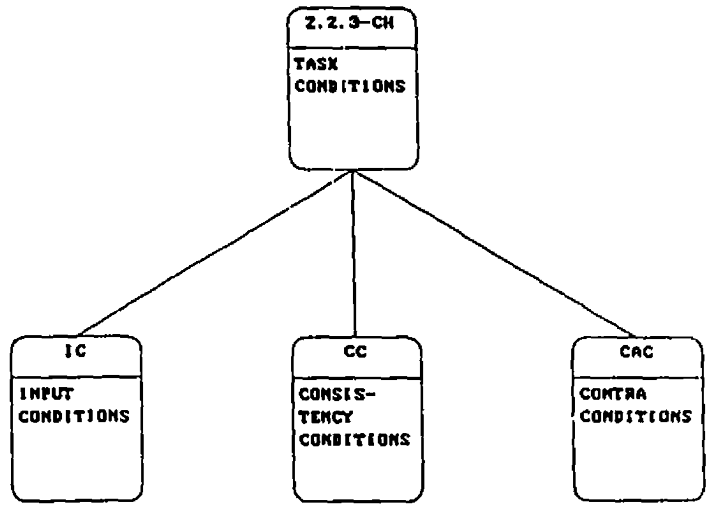


Figure 7.14. Task Condition Class Hierarchy

NAME:	LP.STRUCTURE.EXISTS
SUBCLASS.OF	NIL
INSTANCE.OF	INPUT.CONDITIONS
OWN SLOTS	
status	T
prompt	"Have LP decision variables and constraints been defined in the knowledge base for this problem?"
correction.task	DO.LP.STRUCTURE
METHODS:	
check.condition	
confirm.condition	

Figure 7.15. A Task Condition Object

3.6 Reasoning/control

Task control rules, called by methods in task and condition objects, use forward

```
TASK CONTROL RULES:

TCR1:      IF THE CURRENT TASK IS UNKNOWN
                THEN THE CURRENT TASK OF TASK.CONTROLLER IS THE
                FIRST VALUE IN THE AGENDA LIST IN THE
                TASK.CONTROLLER

TCR2:      IF ?T IS THE CURRENT.TASK
                AND THE CONFIRMED.STATUS OF ?T IS UNKNOWN
           THEN CONFIRM ?T

TCR3:      IF ?T IS THE CURRENT TASK
                AND THE CONFIRMED.STATUS OF ?T IS T
                AND THE CONDITION.STATUS OF ?T IS UNKNOWN
           THEN CHECK THE CONDITIONS OF ?T

TCR4:      IF ?T IS THE CURRENT.TASK
                AND THE CONDITION.STATUS OF ?T IS T
                AND THE CONFIRMED.STATUS OF ?T IS T
                AND THERE ARE SUBTASKS OF ?T
           THEN ADD THE SUBTASKS OF ?T TO THE AGENDA
                AND THE CURRENT TASK IS THE FIRST SUBTASK

TCR5:      IF ?T IS THE CURRENT TASK
                AND THE CONDITION.STATUS OF ?T IS T
                AND THE CONFIRMED.STATUS OF ?T IS T
                AND THERE ARE NO SUBTASKS OF ?T
           THEN EXECUTE ?T

TCR6:      IF ?T IS THE CURRENT TASK
                AND THE CONDITION.STATUS OF ?T IS T
                AND THE CONFIRMED.STATUS OF ?T IS T
                AND ?T HAS SUBTASKS
                AND THE PERFORMED.STATUS OF ALL SUBTASKS
                OF ?T IS DONE
           THEN EXECUTE ?T
```

Figure 7.16. Task Control Rules

chaining to identify the current task at each pause in the debt planning process, based on the candidate task agenda, user approval and satisfaction of prerequisite conditions. Upon selection, the current task is executed and the task identification process repeated. This scheduling/selection/execution loop can be stopped and restarted at any time as long as the agenda, current task and status variables are stored in the knowledge base. Because the detailed knowledge driving the reasoning process is represented in objects rather than in many rules, the process can be handled by a small rule set; Figure 7.16 paraphrases these rules.

Task conditions can be checked by either methods or rule sets called by messages in condition objects. Task condition rules operate by backward chaining to determine the condition status based on knowledge base conditions.

3.7 Contributions of this design

This approach allows significant flexibility in the system control component as well as its eventual extension into a more active planning assistant. As it now stands, it is designed to suggest the appropriate task, explain the reason for the task, check that preconditions are satisfied and execute the task; extensions might include modification of task order based on modelling results (which occurs in a limited way in the parametric and sensitivity analysis subsystem) and active questioning of user requests when they do not seem to fit accepted conventions. Further requirements in this area will be identified by observation and analysis of actual system use.

4. System support implementation status

For the initial system prototype, the following system support functions have been implemented: the user interface (main display screen, menus, object maintenance including form-based input); output tables; output presentation in table and graph format and task management for simulation modelling and projection. Their operation is included in the illustrative planning session in Chapter 10.

8
User Support Design

Although human intermediaries perform a wide variety of tasks to assist managers in using models, this research focussed on four user assistance tasks: (a) optimization model simplification based on the *expected value of perfect information* (EVPI), as explained in Chapter 5; (b) parametric analysis of optimization model results; (c) qualitative explanation of the causes of simulation and cash flow projection results and (d) analysis of the impact on simulation and cash flow projection results of varying the key factors identified as causes of these results. (Many other such tasks should, of course, be added to the system for it to begin to truly replace a human intermediary; one example which is a promising candidate is explanation of optimization model results by linking the ANALYZE program (Greenberg 1987*a,b*, 1988, 1989; Greenberg and Lundgren 1989, Greenberg and Murphy 1989) to MIDAS.) This chapter discusses the initially chosen tasks, presenting the rationale for choosing each as important for this decision situation and then presenting its functional requirements. A common design approach is then proposed to handle all four tasks as well as possible future extensions to MIDAS user support capabilities. The chapter concludes with comments on the research and practical significance of the proposed design and suggests ongoing research in the largely unexplored area of complex financial result analysis and explanation.

1. Task descriptions and functional requirements

1.1 EVPI-based model modification

As noted in Chapter 5, MIDAS debt portfolio models are often too large to be easily manipulated and rerun many times. This difficulty is partly handled by 'intelligent' reduction in model size after it is first solved, based on expected values of perfect information returned by the model solver for each node in the underlying rate tree.

This *model simplification process* must accomplish the following functions:

1. Identification of rate tree branches to be aggregated without significantly changing model results.

2. Aggregation of identified branches in the rate tree by replacing them with single branches containing probability-weighted average rates.

3. Modification of the problem description to reflect the new set of rate scenarios in the altered rate event tree.

4. Modification of the optimization model to reflect the new problem description.

5. Re-solution of the optimization model and storage of new results in the knowledge base.

These functions require knowledge of the impact of EVPI analysis on rate event trees, the underlying problem representation in the knowledge base and procedures for model modification and reruns.

1.2 Parametric analysis

Parametric analysis helps users of an optimization model better understand model behaviour by systematically varying key constraint parameter values, rerunning the model and plotting corresponding variations in the model's objective function value. As noted in Chapter 2, parametric analysis of model results is a standard feature of DSSs which rely on optimization models.

Within MIDAS, parametric analysis must be applied to the constraint on maximum cash outflows for debt service per period. This constraint, as noted in Chapter 5, serves to limit downside risk in a problem situation which theoretically should be solved with an objective containing both a (linear) return and a (quadratic or higher order) risk term. Varying the maximum debt service constraint value and plotting the resulting objective (ending debt portfolio) value gives a curve showing the risk/ending debt value tradeoff for a particular problem.

Parametric analysis might also be used at the user's request to investigate the influence of other constraints on the debt portfolio ending value.

Functional requirements for parametric analysis include the following:

1. Suggestion to the user for further analysis based on varying the maximum debt service constraint.
2. Acceptance of user-initiated parametric analysis requests.
3. Model manipulation (varying of parameter values, rerunning the optimization and storing results) on user request or approval of suggested analysis.

These functions require knowledge of each key factor (including the maximum debt service constraint right hand side) which may be varied at system or user request, together with procedures for varying the factor, modifying and rerunning the model and storing and reporting analysis results.

1.3 Explanation of simulation and cash flow projection results

Model and result explanation and interpretation are identified as key intermediary functions by DSS researchers, as noted in Chapter 2. Explanation can take a number of forms, notably:

- definition of domain or model concepts
- qualitative description of variable requirements and impact on model results
- qualitative description of model relationships and calculations
- detailed quantitative tracing of model calculations
- qualitative interpretation of model results in domain terms or other terms understandable to users
- qualitative causal explanations (partial traces) of model results.

Several of the above types of explanation were eliminated from the initial system design for either practical or research-oriented reasons. First, the intended system users were domain experts and did not appear to require detailed definitions of domain concepts such as 'callable bond' or 'minimum issue size' which are embedded in the system. Second, detailed calculation tracing of simulation and cash flow projection results was found to require significant system overhead and development time; users are satisfied in the short run with the calculation trail provided by detailed output tables. Third, explanation of optimization processes and results is undoubtedly needed for unassisted use of these complex models, but it is the subject of extensive research elsewhere (Greenberg 1987*a*,*b*, 1988, 1989; Greenberg and Lundgren 1989, Greenberg and Murphy 1989); work of this type could be incorporated into MIDAS in the longer term.

MIDAS' explanation capability as initially designed is intended to provide simple, high-level, predefined descriptions to aid user understanding of analysis purposes and results. (More complex explanations based on a dynamically-determined qualitative model of the analysis process as in Bouwman (1983) would be a natural extension of the system but are beyond the scope of this project for reasons discussed later in this chapter.) The following explanations are provided for simulation and cash flow projection models:

1. A qualitative description of model and sensitivity analysis steps and results in terms that are readily understandable by system users.

2. A high-level causal explanation of these results in terms of major determining factors and their impacts.

These two types of explanation are intended to provide some insight into borrowing plan performance as demonstrated by the simulation and cash flow projection models. They are also meant to help the user understand the significance of particular complex model results and to begin to trust the models that produce them to a greater degree than would be probable with no result explanation.

Explanation content varies with the particular model producing the results to be explained, the characteristics of the particular problem being modelled and patterns in model and sensitivity analysis results. Producing these explanations therefore requires knowledge of the following types:

1. Major *result types*, such as stochastic portfolio optimization and multi-scenario simulation, single-scenario simulation and single-scenario cash flow projection for portfolios and individual debts.
2. A qualitative description of the model determining each result type.
3. The specific contents of each result type.
4. Qualitative interpretations of these contents.
5. Potential key factors for each result type, such as the single largest debt, long-term high-rate debts, and high-rate time periods.
6. Procedures for determining relevant key factors for a set of specific model results.
7. Qualitative explanations of key factor impact.

This knowledge is both factual and procedural, involving in-depth knowledge of the composition and underlying causes of types of model results, and knowledge of procedures for analyzing results to determine which causes apply in particular cases.

1.4 Analysis of key-factor variation on simulation and cash flow projection results

The sensitivity of a borrowing plan to movements in future rates is measured by the results of the plan simulation process, which projects plan results repeatedly with random rate variations. To complement these, however, the user often wishes to test the impact on simulation results of other assumptions or plan modifications such as a change in the timing, term or market for a major proposed debt. The *key-factor impact analysis* feature is designed to manipulate a simulation or cash

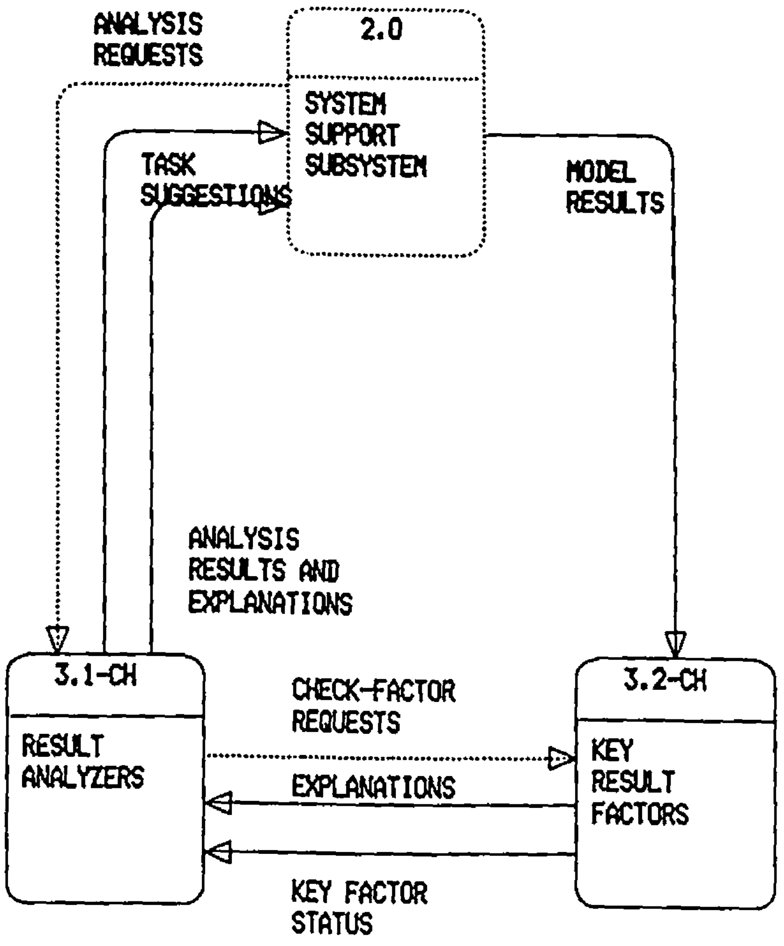


Figure 8.1. User Support Subsystem

flow projection model to determine the sensitivity of its results to the key plan factors identified as part of the explanation process. Specifically, it is designed to accomplish the following functions, once key causal factors have been identified:

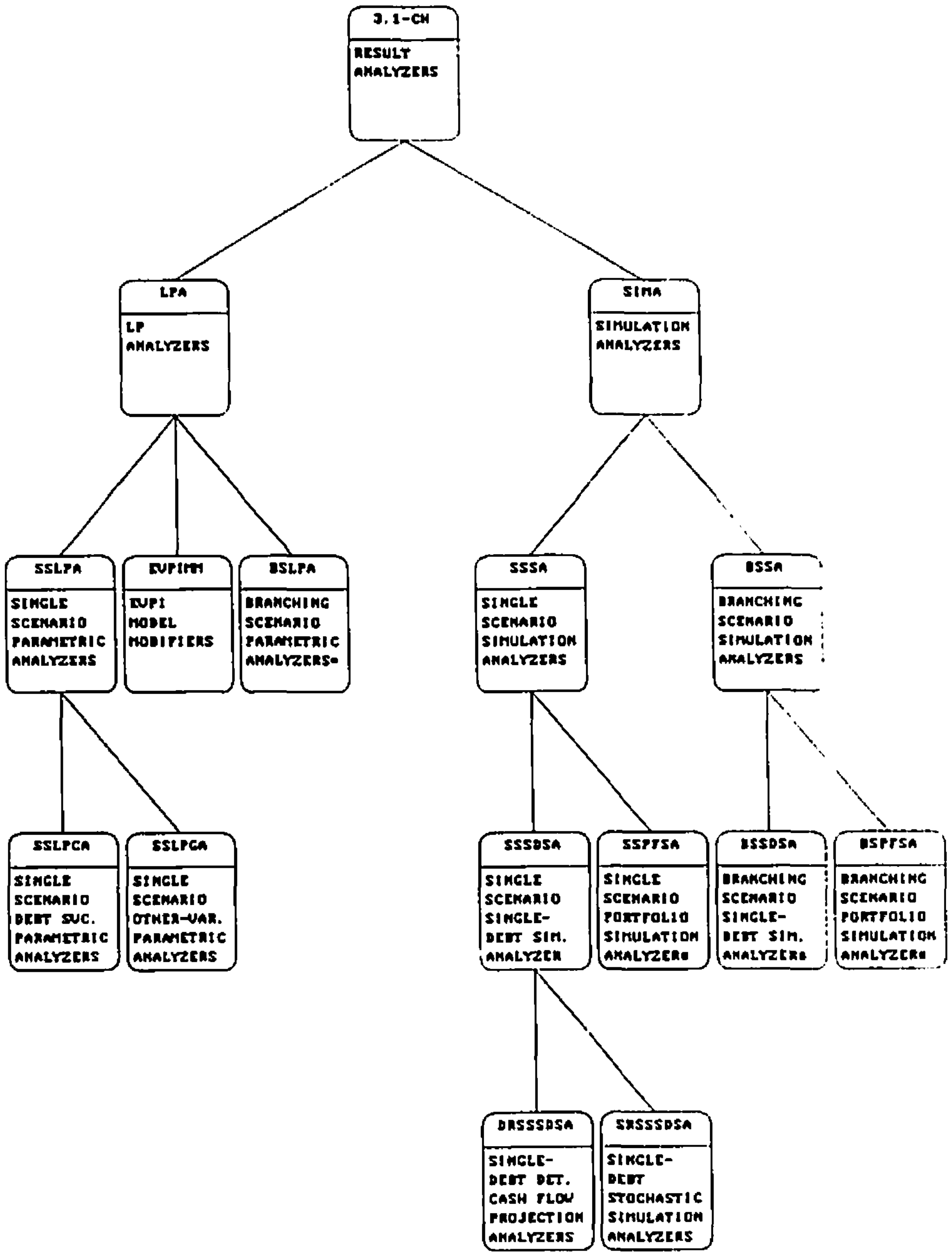
1. Identification and suggestion to the user of further analysis steps based on the identified key factors.
2. Acceptance of user-initiated analysis requests.
3. Model manipulation (varying of plan specifications or parameter values, rerunning the model and storing results) on user request or approval of suggested analyses.

These functions require knowledge of the further analysis tasks that are appropriate once key factors have been identified, as well as knowledge of procedures for carrying out the analysis tasks. As for explanation, this knowledge has both factual and procedural components.

2. Proposed knowledge representation

Although the four tasks described above appear at first to be quite diverse, they have two common underlying elements. First, all tasks are based on certain key model results; for EVPI analysis and parametric analysis of maximum debt service, these factors (the EVPI for each rate event tree node and the maximum debt service constraint) are known in advance, while for other analyses and explanations the factors are dynamically identified by the system or the user. Second, each task requires the execution of one or more system operations based on the key model results known or identified for the result type.

These similarities among user support functions allow them to be handled by a single design approach, which is used for the proposed user support subsystem, shown in Figure 8.1. Within the subsystem, user support functions follow the data-driven, frame-based approach already seen in system support (Chapter 7). Each


3.1-CH
RESULT ANALYZERS
LPA
LP ANALYZERS
SIMA
SIMULATION ANALYZERS
SSLPA
SINGLE SCENARIO PARAMETRIC ANALYZERS
EVPIMM
EVPI MODEL MODIFIERS
BSLPA
BRANCHING SCENARIO PARAMETRIC ANALYZERS*
SSSA
SINGLE SCENARIO SIMULATION ANALYZERS
BSSA
BRANCHING SCENARIO SIMULATION ANALYZERS
SSLPCA
SINGLE SCENARIO DEBT SVC. PARAMETRIC ANALYZERS
SSLPGA
SINGLE SCENARIO OTHER-VAR. PARAMETRIC ANALYZERS
SSSDSA
SINGLE SCENARIO SINGLE-DEBT SIM. ANALYZER
SSPFSA
SINGLE SCENARIO PORTFOLIO SIMULATION ANALYZER•
BSSDSA
BRANCHING SCENARIO SINGLE-DEBT SIM. ANALYZER•
BSPFSA
BRANCHING SCENARIO PORTFOLIO SIMULATION ANALYZER•
DRSSSDSA
SINGLE-DEBT DET. CASH FLOW PROJECTION ANALYZERS
SRSSSDSA
SINGLE-DEBT STOCHASTIC SIMULATION ANALYZERS

*LINKED TO SUBCLASSES FOR PARAMETRIC ANALYZERS FOR MAXIMUM DEBT SERVICE CONSTRAINT AND FOR OTHER VARIABLES

•LINKED TO SUBCLASSES FOR DETERMINISTIC AND STOCHASTIC SIMULATION ANALYZERS FOR EACH RESULT TYPE

Figure 8.2. Result Analyzer Class Hierarchy

```
NAME:              BRANCHING.RATE.DEBT.SVC.ANALYZER
SUBCLASS.OF:       NIL
INSTANCE.OF:       BRANCHING.SCENARIOS.PAR.ANALYZERS
ATTRIBUTE SLOTS:
    result.type         BRANCHING.RATE.OPTIMIZATION
    result.description  "This set of results comes from the execution of a stochastic
                        optimization model which selects from available debt
                        types to produce an optimal borrowing plan and debt
                        portfolio for meeting cash requirements over the defined
                        planning period. The plan hedges against future rate
                        uncertainty and produces a contingent plan which varies
                        according to future rate scenario."
    result.source       PF23.TABLE
    key.factors.list    (MAX.COST.CONSTRAINT)
    analysis.tasks      (DO.PA.MAX.COST)
EXTERNAL-USE METHODS:
    check.key.factors
    plan.analysis
```

Figure 8.3. A Result Analyzer Example

function is driven by knowledge in *result analyzer objects*, which contain knowledge about result types, including result type descriptions, names of possible or known key factors for explaining the results, names of related analysis tasks and generalized methods for checking key factors and setting up analysis tasks on the task agenda. Each key factor is, in turn, represented by a separate *key factor object*, which contains rule initiators or procedures for determining whether the factor is relevant and important for a particular set of results. The key factor objects also contain explanations and analysis task knowledge for building analysis task sequences once key factors are known.

Analyzer objects are defined by function and result type in a class hierarchy (Figure 8.2), and appropriate analyzers are instantiated when a set of results is to be analyzed. Figure 8.3 shows a result analyzer for a portfolio within a problem defined with a branching probabilistic rate event tree. This analyzer handles parametric analysis based on the maximum cost constraint for the optimization model; the *check.key.factors* method is a generalized one which requests a checking operation for each key factor in the *key.factors.list*, and the *plan.analysis* method adds the analysis task slot value to the current agenda.

Key factor objects are defined in a class hierarchy for potential key factors in each type of results; an appropriate set of key factor objects is instantiated when a particular set of results is to be analyzed. Figure 8.4 illustrates a key factor object which supports the result analyzer for maximum cost parametric analysis for optimization models. In this case the relevance (status) of the factor is known beforehand; if it is not, it is checked by the *check.status* method, which may request LISP procedures or rule-based reasoning, in the object. The *analysis.task* name is added to the list of the same name in the result analyzer during analysis planning.

Explanations generated by these objects are either *result descriptions*, stored by result type in the analyzer objects, or *key factor explanations*, which depend on the result type and on the key factors applicable to particular result sets. The key factor explanations shown in these examples are produced dynamically for individual result sets by the methods or rules that identify key factors and check their status.

3. Reasoning and control

For a given set of model results, the analysis process involves (a) identifying the key factors accounting for the results, and (b) carrying out selected analyses related to these factors. Because extensive research must be done before comprehensive sets of factors and analysis procedures can be defined, specific procedures and rules have not been given for this process. However, outlines are given here for several major types of analysis that are likely to be required. Each process might be handled through LISP procedures or rules, depending on its particular requirements.

```
NAME:               PF23.MAX.COST.SVC.CONSTRAINT


SUBCLASS.OF:        NIL


INSTANCE.OF:        MAX.COST.CONSTRAINT.KFS


ATTRIBUTE SLOTS:
    result.source      PF23.TABLE
    factor.status      T
    status.explanation "The maximum cost constraint replaces the borrower's
                       downside risk objective in creating a borrowing plan.
                       It is always a key factor in determining optimization results."
    impact.explanation"The ending value of a portfolio determined by
                       a borrowing plan normally increases the lower the cost
                       constraint and decreases the higher the cost constraint.
                       This is because a looser constraint allows the borrower
                       to take greater advantages of short-term lower rates
                       while it increases the risk that refinancing will have to
                       take place at higher rates later."
    analysis.tasks     (DO.PA.MAX.COST)


EXTERNAL-USE METHODS:
    check.status
    analyze
```

Figure 8.4. A Key Factor Object Example

3.1 EVPI-based optimization model simplification

The EVPI-based model modification process begins with simplification of the rate event tree by averaging low-value branches and collapsing them within a copy of the tree; a new set of worlds is then created reflecting the new scenarios and the model recreated and re-solved as described in Chapter 6. Res 'ts are stored in the new world set and reported as for the original model.

3.2 Parametric analysis

For parametric analysis of optimization model results, the factor to be varied is the maximum cost constraint right-hand side value unless specified otherwise by the user. Once this factor is known, the process consists of: (a) identifying the upper and lower limits of variation for the factor; (b) identifying specific values for the varied factor within this range, such as two values above and two below the value in the original model; (c) for each new value, modifying only relevant model input items; (d) rerunning the model with modified input and storing the resulting objective end value; and (e) presenting the parametric analysis results as a line plot. A structure chart for this process is shown in Figure 8.5.

3.3 Key factor identification

For the simulation model, key factors must be identified prior to explaining or further analyzing results. Reasoning for the key factor identification process is done by a method which (a) identifies and instantiates the appropriate analysis object, (b) calls factor-specific methods to check all candidate factors for relevance and importance, and (c) builds a task list based on factor status and estimated impact. Once the task list is built, the list is placed at the top of the task agenda and executed consecutively by the system control subsystem.

3.4 Key-factor impact analysis

This analysis proceeds by holding all factors except the identified one constant,

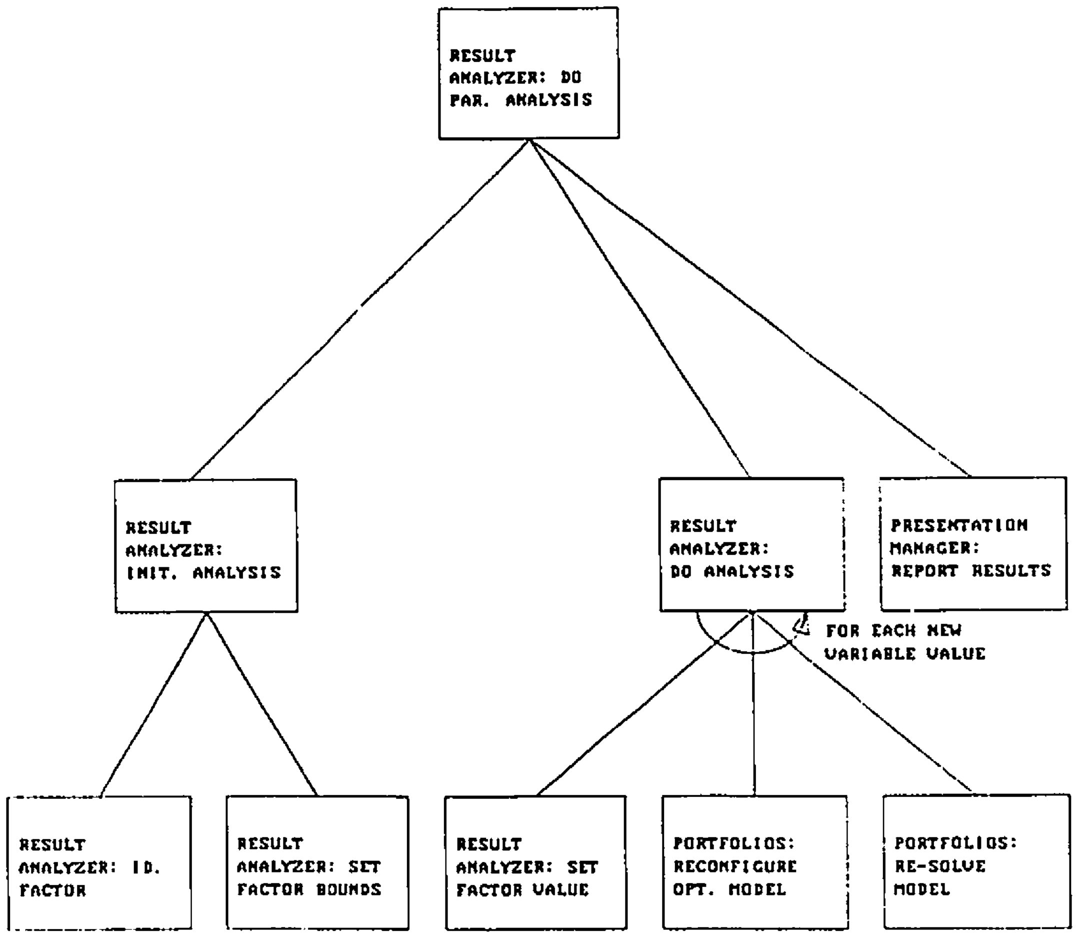

RESULT ANALYZER: DO PAR. ANALYSIS
RESULT ANALYZER: INIT. ANALYSIS
RESULT ANALYZER: DO ANALYSIS
PRESENTATION MANAGER: REPORT RESULTS
FOR EACH NEW VARIABLE VALUE
RESULT ANALYZER: ID. FACTOR
RESULT ANALYZER: SET FACTOR BOUNDS
RESULT ANALYZER: SET FACTOR VALUE
PORTFOLIOS: RECONFIGURE OPT. MODEL
PORTFOLIOS: RE-SOLVE MODEL

Figure 8.5. Parametric Analysis Control

varying it in prespecified intervals and re-simulating only the affected debt or debts in the affected time periods, and re-totalling the portfolio, in a new KEEWorld.

For a portfolio, any changes in the portfolio amount relative to cash requirements are handled simply by increases or decreases in the cash deficit or surplus balance—parameters in the portfolio other than the deficit or surplus and the factor being varied are not adjusted.

3.5 Result descriptions and explanations

Descriptions and explanations are presented by *text display managers* in MIDAS' system support component (Chapter 7). These use LISP methods to retrieve explanations, format them into text and present them in expl[illegible] windows.

4. Contributions and limitations of this design

As for other MIDAS components, the proposed result analysis design illustrates the advantages of explicit knowledge representation and generalized reasoning. It also presents an approach toward building intermediary assistance into the system in a way which integrates in a straightforward manner with other system components. In addition, the proposed design appears extendable to other analyses that would be required in a full system, such as identification of infeasibility conditions for optimization, analysis plan comparison, plan evaluation and choice among competing plans.

The explanation design is based on a simple diagnosis model which, once the appropriate result type and key factors are identified, has explanation text passages ready for use. For any one result type and its associated analysis frame, its flexibility is limited to including or excluding possible key factors from an explanation based on their identification as relevant or important. However, the scheme does provide the most basic explanations needed by system users. Moreover, it could be expanded by (a) adding rule-based reasoning for more flexible key factor evaluation, and (b) adding analysis frames that distinguish among result types on a finer basis than simply by model type in a CENTAUR-like (Aikens 1983) diagnosis process.

The greatest benefit of this aspect of the project from a research viewpoint is its identification of issues which could benefit from further research. First, understanding these complex simulation and cash flow models undoubtedly requires many types of explanations of varying degrees of detail. However, researchers do not appear to have identified the content or scope of these explanations apart from detailed calculation traces or rudimentary qualitative models, and they were not obvious during the limited use MIDAS received during this project. Additional research observing and analyzing the explanations requested by users and given by human intermediaries would be a first step in defining detailed explanation requirements for complex models of the type used in MIDAS.

Second, explaining the significance of simulation and cash flow model results in domain terms appears to require some assessment of result importance relative to either borrower norms or industry standards. (For example, one way of deciding which output results to highlight would relate performance measures to borrower expectations, market performance or the performance of other actual or hypothetical debt plans.) The Corporation norms or standards for debt portfolio planning can be better articulated after they have gained some experience and expertise with this planning approach, which requires use of the system for some time. Market and industry standards may exist but would need to be further investigated before incorporation into the system.

Finally, qualitative model and output explanations are likely to require more knowledge than that provided by a diagnose-and-explain process mapping a predetermined set of result patterns into predefined explanation templates. Rather, they are likely to require qualitative representations of model relationships which can be instantiated more flexibly for particular models and sets of results, as illustrated in Bouwman (1983) and in research on the qualitative modelling of physical systems

(Bobrow 1984). Construction of dynamic 'models of the models' in MIDAS appears to be significantly more difficult than for the simple financial statements Bouwman investigated, and further work is needed to identify the concepts and knowledge structures needed to build simplified but meaningful descriptions and to identify ways of implementing them that do not add unacceptable overhead to the system.

In summary, MIDAS' simulation explanation component illustrates a straightforward way to incorporate rudimentary explanation facilities into model result representations. Further research will be required to build truly 'intelligent' explanation capable of responding to the full range of explanation content, domain conditions and model relationships which should be taken into account; however, this area is a fertile one for future research.

9

Development History and Experience

Many aspects of this project, including the system concept, modelling approach, user assistance requirements, design approach and implementation techniques were unexplored by previous management science, DSS and AI researchers. The experience gained in attempting to conceptualize, design and implement the system therefore suggests many lessons about successful approaches to solving the problems involved. This chapter outlines MIDAS' development history, identifying and explaining key design and development decisions; it also discusses successes achieved, strengths and weaknesses of the development tools, difficulties encountered and significant unexplored research and technical issues identified as the project progressed.

1. Problem and system scope

The MIDAS project was motivated initially by a combination of research and practical goals. Research motivations included (a) the desire to explore certain capabilities of AI for modelling and other forms of decision support, and (b) the desire to test on corporate problems the stochastic optimization tools now being developed at the Dalhousie School of Business Administration. The practical motivation was the need to improve Corporation borrowing decisions, ultimately saving significant debt cost for the company; the Corporation viewed the project as a way of adding to their expertise rather than encoding their present expertise into a system. Corporation staff also expressed a need for rate forecasting assistance, and initial interviews with debt underwriters and advisors indicated that much of the actual borrowing decision process was based on heuristics which could be embodied in a rule-based system.

Final project scope was based on a combination of Corporation needs, research interests and the availability of expertise. The central system functions were determined to be modelling and modelling support rather than heuristic planning; rate forecasting was eliminated as already available from Corporation financial advisors. (Planning heuristics and forecasting assistance could, however, be added as later system extensions.) The project's model base and planning approach were developed to provide the simulation required by the Corporation and to guide and extend it with stochastic optimization; consideration of the complementary roles of the two types of models led to the hierarchical debt planning approach using optimization, heuristic plan refinement and simulation as described in Chapter 4. User assistance selected for immediate study included task scheduling, parametric and sensitivity analysis and simulation explanation; these were chosen based on prior experience of the principal researchers acting as modelling experts and advising clients in similar situations.

2. Tool selection

The variety of functions envisioned for MIDAS indicated a need for several types of knowledge representation and control/ reasoning approaches. Powerful hardware was also needed to handle the anticipated size of the domain representation and the stochastic portfolio models, ruling out the PC-based systems available in early 1986 when the project was first considered. The Explorer hardware with KEE software met these basic requirements as well as or better than other tools available at the time. The Explorer offered 4mb of RAM, 200mb of hard disk storage, virtual memory and a full-scale LISP development and debugging environment. KEE included frames and object-oriented programming, KEEWorlds for representing multiple problem states, demons to automatically maintain slot values, rules with multiple control strategies, and many other features giving great flexibility in system design. The development system was made available to Dalhousie as a research grant; it was therefore selected in spite of certain weaknesses, to be discussed later.

3. Development approach and prototype history

As mentioned in the introduction to this dissertation, an experimental, evolutionary prototyping approach to development was used. This allowed the researchers to begin with relatively simple problems, gain experience with the design methodology and development tools, and progress to more complex system functions. This approach turned out to be valid, and major discoveries at each prototype stage influenced and changed the design as it progressed.

The project extended over four years and directly involved, as designers and developers, nine people, primarily on a part-time basis. Athough detailed time records were not kept, total development (system learning time, detailed design, coding, testing and debugging, excluding work on the LP solver) is estimated to have taken between three and four person-years.

Prototype stages were as follows:

1. *Single-debt simulation prototype.* Prototype 1 produced a simple financial instrument hierarchy and single-debt deterministic (spreadsheet-like) simulation. Only basic model objects were included. These contained all definitional slots, object maintenance methods, simulation calculation methods, output slots and output presentation methods. Cash flow calculation was done line-by-line, the simplest approach for single debts. The initial prototype, covering bonds only, was begun in January 1987, and completed in approximately two months; it was extended to cover additional debt types between April and October 1987.

2. *Portfolio simulation prototype.* This extension of the first protytpe included a full debt hierarchy including all debt types currently used by the Corporation, simulated either individually or in portfolios. It used the original financial instrument object structure and added portfolios (composite model objects). Cash flow calculations were changed to operate period-by-period in order to allow for dynamic decisions within the simulation based on current portfolio status. This prototype was developed between April and December 1987.

3. *Stochastic portfolio simulation.* Prototype 3 extended Prototype 2 to include simulation with random rate generation around a single rate path, more extensive output displays and graphic display of yield curves. This was completed in the spring of 1988 and was extensively tested and verified by Corporation staff.

   Prototype 3 was used and approved by Corporation staff; however, it could only be run on the Explorer at Dalhousie University. The less costly MicroExplorer had recently been announced and efforts to obtain one for implementation in the Corporation offices began in mid-1988. A MicroExplorer was acquired at Dalhousie in October 1988, and the system needed only modifications in file path names and window size specifications to be ported to it. After long administrative delays not related to the project, hardware was acquired by the Corporation and the system was tested and installed on it in the summer of 1989.

4. *Development of the communications interface and optimization solver.* During the fall of 1989, LISP methods were developed for connecting with the optimization solver via Ethernet. Between January and June of 1990, the optimization solver was run on typical test problems and extended to produce EVPI data as output.

5. *Extended simulation prototype.* Between August 1989 and June 1990, a new prototype was constructed to illustrate further MIDAS design concepts. This system incorporated greatly extended expertise through the use of frame-based representation of abstract concepts with generalized methods and rules; as described in several chapters of this dissertation, the technique was applied to output management and task management. This system excluded certain debt types found in previous versions in order to reduce the complexity of the prototype for testing the newly-implemented functions. This system was tested on the test data provided on microfiche with this dissertation. It produced the planning session in Chapter 10.

6. *On-going work.* During the summer of 1990, work is progressing on graphic result

presentation and on the LP support component of the modelling subsystem.

4. Knowledge base evolution

Knowledge representation structures and control strategies used in MIDAS evolved as experience was gained with KEE and as problem and model structures became more clearly apparent. The overall evolution was one of greater use of abstract knowledge and more generalized, data-driven reasoning and control with each prototype.

4.1 Model object structure

Initially the guiding concept defining model objects (debts, investments and portfolios) was that of objects that maintain, model and report on themselves. Standard protocols (method names and parameters) were to be used regardless of model object class, and generic high-level control methods were to determine maintenance, modelling or output method execution sequence.

Object maintenance and modelling functions were retained in model objects in the final prototype, although they were streamlined by the use of generic methods attached to multiple slots in several situations such as add, change and delete. (Input form generation and processing were specific to each object class.) Output storage and presentation were, however, removed from model objects and handled separately by system support objects for reasons cited above and in Chapter 6.

Model support objects were defined from client and financial advisor descriptions of the domain. They were retained through all prototypes and continue to provide an intuitively clear domain model as a base for both model organization and user communication about the system.

4.2 LP support

The first outline for generating optimization input from the knowledge base anticipated coding all coefficient sources and input formats directly into the LISP methods which build the standard input files. Changes in the model's structure or specification would therefore require (relatively difficult) changes in method code rather than in (less complex) objects and slot values, a need that had largely been avoided for other parts of the system that were likely to require modification in the future. The current data-driven approach captures knowledge of optimization model structure in specifier objects and their slot values, so that the structure is more easily understood and maintained through modifications to explicit data values in the knowledge base. The current design also allows more flexible and automatic reformulation of optimization models in response to problem statement changes.

4.3 User interface and system support

The user interface is designed for clear interaction and multiple presentation modes; however, it turned out to be the most difficult aspect of the system to implement due to the low level of business presentation support in KEE and the Explorer. In all prototypes but the last, output presentations were handcrafted to fit individual display requirements.

Output table and presentation manager design came about after analysis of the fundamental structure of the repetitive code being used through the system for output management and presentation. This led to the data-driven approach described in Chapter 7. It is estimated that implementation of these techniques reduced total system size by approximately 30% and also made possible the relatively straightforward implementation of graphic output presentation.

4.4 Task control and user support

The data-driven approach is also the basis for the task control and user support component designs, which are intended to provide frameworks for implementing a variety of types of assistance to novice users of the system. As noted in earlier chapters, the task control component operates successfully, and indications are that success is likely with the user assistance subsystem in view of the success of the approach for other functions. These designs will undoubtedly evolve as they are implemented and used and as user requirements are better understood.

As discussed in Chapter 8, the attempt to design result analysis and explanation capabilities for the MIDAS models has led to identification of several major issues for further research.

5. The choice of rules versus methods

Of the many knowledge representation and reasoning tools provided by KEE, object-oriented programming and integrated rule-based reasoning were clearly the most useful for this project. Object-oriented programming, or the incorporation of procedures and reasoning into methods rather than rules, was used more extensively than rule-based reasoning for two reasons. First, many functions related to modelling and data handling involved well-defined algorithms and did not vary greatly with the data under consideration; variations due to financial instrument type or output type were handled by varying method definitions by object class. Second, some generalized reasoning sequences and procedures such as optimization model construction could have been represented as rule sets; however, the additional overhead and complexity involved was judged to be unacceptable at this stage in the system's development. (Rule-based linear program formulation has been illustrated in other systems, however, as noted in Chapter 2.) In the present design, then, rules are used in situations which involve primarily symbolic reasoning, which require relatively little knowledge-base object manipulation or calculation, and which clearly emulate human-expert functions for model management and assistance. The addition of facilities for error-checking and knowledge base consistency maintenance, both of which are necessary for commercial use of the system, should offer additional opportunities for the efficient use of rules.

6. Hardware/software advantages and difficulties

The Explorer/KEE development system was discovered to have both advantages and disadvantages for a project of this type. Its major advantages include the overall flexibility and modularity provided by object-oriented software design and construction; the extensive LISP editing and debugging environment on the Explorer, and the flexibility of KEE's integrated multiple knowledge representation tools. Also, LISP itself eliminates the need to consider memory management, data types and many other low-level details in designing software.

KEE's object-oriented programming facility proved quite flexible and able to handle any function in the design. KEE's Rulesystem also proved adequate for the areas in which it was used. KEEWorlds were not used in the implementation, but this facility is designed to handle multiple plans and scenarios and so will be used for stochastic modelling and multiple-plan analysis in future system versions.

Two KEE tools were tried and found to create difficulties within the system. First, demons (called *active values* in KEE) were initially used to automatically maintain consistent slot values within debts, but they were largely eliminated from later prototypes because they greatly complicated knowledge base changes and method debugging. They were replaced by consistency calculations within input methods. Second, multiple method inheritance was tried as a way of building calculations for bonds with multiple parents (for example, callable bonds with sinking funds); however, KEE's way of building single methods out of inherited code from multiple parents turned out to be quite cumbersome. It was avoided by checking debt parent classes within calculation methods and altering calculations accordingly.

Several more general characteristics of the development environment made system implementation more difficult than originally anticipated. The first of these was the extremely steep and long learning curve for KEE and the Explorer, which made it especially difficult to productively use student programming assistants. A second difficulty was the lack of tools for easily constructing the user interface;

KEE is oriented mainly toward industrial applications and does not provide easily defined forms, reports or business graphics. Third, the system was not ideal for development of financial models since calculations had to be written in LISP messages with lengthy procedural syntax and high data retrieval overhead; a more suitable tool would include a higher-level language for model definition that would lead to more rapid prototype development than was able to be accomplished with this project. Fourth, its specialized operating system and incompatibility with other systems make both communications among systems and long-term support difficult and costly. Finally, system performance may be a problem on the current hardware when LP support and worlds are implemented and realistic problems attempted, since the large knowledge base and extensive computations which will be required may strain the processing power and virtual memory of the current machines. In the long run, however, new and more powerful hardware and software should address this question.

7. Documentation

One major issue arising during MIDAS design and development was the need for new techniques to document object-oriented systems. Existing structured analysis and design techniques perform well in systems in which data and procedures are logically separated; they do not apply, however, when the two are integrated as in KEE objects. Ideally, what is needed are techniques for clear documentation of high-level design, object structures and method control paths. Object diagrams that are common in the literature (for examples, see Pressman (1987)) do not appear capable of clearly describing a complex system such as MIDAS; Coad and Yourdon (1990) outline a multi-level approach to documenting object-oriented system requirements which may prove useful in the future, but it was published near the end of this project and is not supported by currently-available CASE tools. The diagrams and system documentation in this dissertation use modifications of dataflow diagrams and structure charts produced by the Excelerator CASE tool (Index Technologies

1989).

An additional facility that would be invaluable in a tool such as KEE is an automatic documentation generator; as part of this project, a method-call trace was developed which begins to solve that particular problem.

8. Corporation involvement with the system

From the Corporation's viewpoint, this project has had little impact in the short run. Two major reasons for this are (a) the difficulty of producing comprehensive, supportable code in reasonable time, and (b) the untimely death of its original champion, the then Director of Management Information Systems, in mid-1986. However, the Corporation did provide full-time development participation from a senior expert systems staff member for approximately six months. Subsequent wavering MIS support, administrative delays, development delays, lack of technical support for converting prototypes into reliable system versions, and numerous personnel and organizational changes within the Corporation, have kept the system from being regularly used by the Treasury staff.

However, indications are that the project is still viewed as important by the Corporation. Treasury and MIS staff have recently confirmed their support for the project, committed a temporary staff member to full-time support and development of a production version for several months beginning in June 1990, and named a permanent member of the MIS department to handle long-term support for the system. Future research plans are to obtain feedback from Treasury staff members and to carry out experiments observing system use in order to help identify appropriate extensions on which to focus further research.

From a research perspective, the project has suggested both design principles and implementation techniques that should extend present financial DSS capabilities. These have been identified in previous chapters and are summarized in the concluding chapter of this dissertation.

10
An Illustrative MIDAS Session

The system prototype developed as part of this research implements most aspects of simulation modelling and cash flow projection for financial instruments and portfolios for a single rate scenario, together with the system support components (the user interface, task management and output management) described in Chapter 7. Although borrowing actions are not yet able to build debt objects for simulation and projection, future borrowing plans can be tested by directly creating portfolios of debt objects with issue dates during the planning period. Heuristic plan refinement has been implemented to operate on user-defined borrowing actions. Major functions yet to be implemented include optimization modelling (although the required communications links, the solver and EVPI analysis are in operation), multiple branching scenarios using KEEWorlds, and user support as outlined in Chapter 8.

In terms of the task sequence in Figure 7.13, tasks numbering 1 through 3 (problem description), 7 (plan refinement), 8 (simulation or cash flow projection), 11 and 12 (save problem and end session) can be carried out by the prototype with only minor omissions; tasks 4 through 6, related to optimization and 9 and 10 (result analysis and explanation) are not included.

This chapter describes a session using the system prototype for simulation and cash flow projection for a small portfolio. It generally follows the task sequence in Figure 7.13, showing and explaining the system's dialogue with the user through screen images and text. It illustrates the system's approach to the simulation aspect of debt management decision support; the tasks not included in the prototype are designed to operate and interact with the user in a similar manner.

The following conventions are used in the session description:

1. Prompts and comments produced by the system are shown in `typewriter` font.
2. Typed user replies are shown in CAPITALS.
3. The main user interface, input forms and output displays are shown through screen dumps.
4. Comments in *italics* clarify the description as necessary.

1. Problem space initialization

The assisted analysis process is started by left-clicking on the `BEGIN ASSISTED ANALYSIS` *option in the main menu.*

```
First you need to choose a problem to work on and set some over-
all work session parameters.  OK to begin (Y/N)?
```

Y

```
Enter the name of the current problem.  Use 'NEW' to begin a new
problem.
```

DEMO

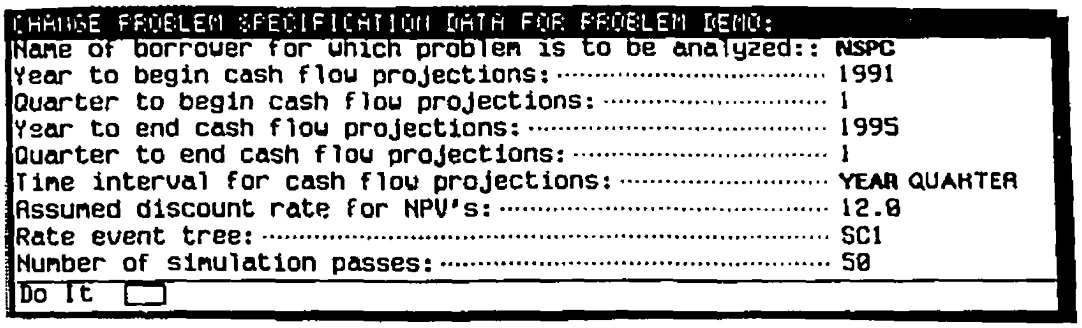
The 'DEMO' *problem space, which contains a partially specified problem including future rate scenarios and debts, is loaded. If* 'NEW' *is chosen, a knowledge base containing only financial markets, a borrower and the borrower's existing debts is loaded; this problem space is then renamed and modified to represent a new problem. If a name is entered for which there is no knowledge base, an empty knowledge base is created which must be fully specified by the user.*

```
Problem space DEMO has been successfully loaded/created.

Do you wish to change the problem specification data for this
problem (Y/N)?
```

WHY

This will allow you to change analysis parameters including the starting date, ending date, time interval, rate forecast name and number of simulation passes. Do you wish to do so (Y/N)?

Y



```
CHANGE PROBLEM SPECIFICATION DATA FOR PROBLEM DEMO:
Name of borrower for which problem is to be analyzed:: NSPC
Year to begin cash flow projections:.................. 1991
Quarter to begin cash flow projections:............... 1
Year to end cash flow projections:.................... 1995
Quarter to end cash flow projections:................. 1
Time interval for cash flow projections:.............. YEAR QUARTER
Assumed discount rate for NPV's:...................... 12.0
Rate event tree:...................................... SC1
Number of simulation passes:.......................... 50
Do It
```

What type of prompts would you prefer (LONG or SHORT) (use LONG if you are not familiar with the analysis process and would like help as you go along)?

Long prompts guide a less experienced user, while short prompts give quick orientation messages to a user who is familiar with system operations.

LONG

What type of analysis do you wish to do: (1) simulate a portfolio; (2) project cash flows for a portfolio; (3) simulate a debt; (4) project cash flows for a debt?

The type of analysis to be done affects knowledge and parameter requirements for the problem. Simulation or projection for a portfolio requires that all debts in the portfolio be defined and that borrowing actions be specified for calls and planned short-term borrowing and investment; simulation or projection of an individual debt requires only that the debt be defined in the problem space. The full MIDAS system *as designed will also contain options to BUILD A BORROWING PLAN using the full optimization/heuristic refinement/simulation process, and SIMULATE or PROJECT CASH FLOWS for a BORROWING PLAN, in which modelling will be based on a full set of borrowing actions rather than on user-defined debt objects.*

1

2. Review of background knowledge

```
Before the problem can be described, you should
review the specifications of financial markets,
borrower requirements and future rates for the
problem.  OK (Y to confirm, S to skip, N to select
an alternate task)?
```

If N, *the system displays for user choice names of appropriate alternative tasks which vary with the current task but generally include returning to the previous task, returning to the session start, exiting and saving the problem space and exiting without saving.*

Y

2.1 Market knowledge

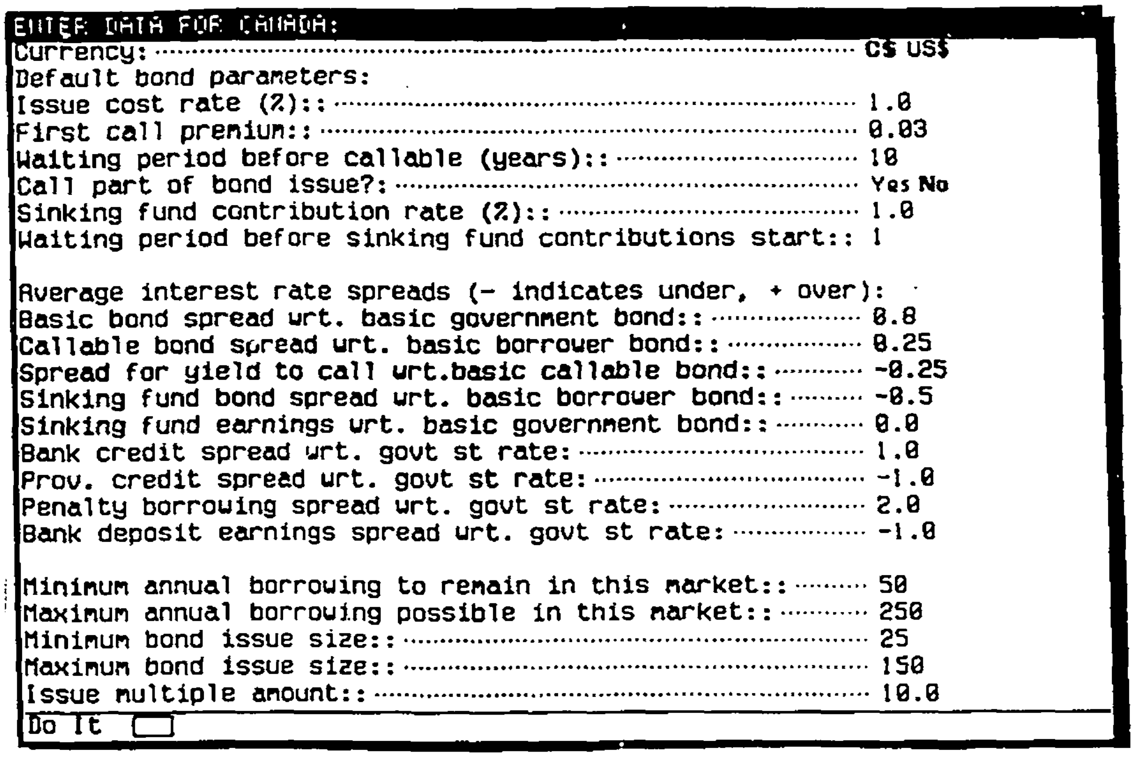
```
The following markets are defined for the problem:
CANADA, US. Do you wish to review them (Y to
confirm, S to skip, N to select an alternate task)?
```

Y

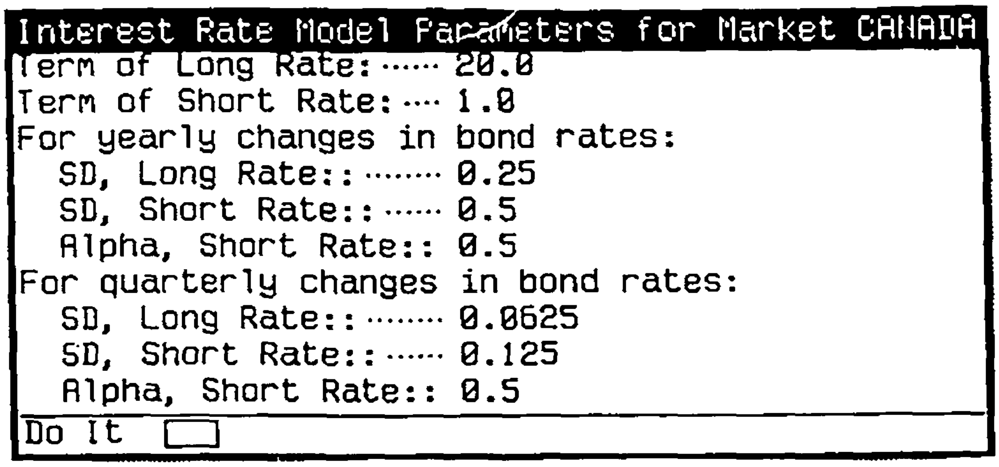
```
Choose a market to review (CANADA/ US).
```

CANADA

Please check that the parameters for CANADA are correct and change them if necessary.



```
ENTER DATA FOR CANADA:
Currency: ............................................. C$ US$
Default bond parameters:
Issue cost rate (%):: ................................. 1.0
First call premium:: .................................. 0.03
Waiting period before callable (years):: .............. 10
Call part of bond issue?: ............................. Yes No
Sinking fund contribution rate (%):: .................. 1.0
Waiting period before sinking fund contributions start:: 1

Average interest rate spreads (- indicates under, + over):
Basic bond spread wrt. basic government bond:: ........ 0.0
Callable bond spread wrt. basic borrower bond:: ....... 0.25
Spread for yield to call wrt.basic callable bond:: .... -0.25
Sinking fund bond spread wrt. basic borrower bond:: ... -0.5
Sinking fund earnings wrt. basic government bond:: .... 0.0
Bank credit spread wrt. govt st rate: ................. 1.0
Prov. credit spread wrt. govt st rate: ................ -1.0
Penalty borrowing spread wrt. govt st rate: ........... 2.0
Bank deposit earnings spread wrt. govt st rate: ....... -1.0

Minimum annual borrowing to remain in this market:: ... 50
Maximum annual borrowing possible in this market:: .... 250
Minimum bond issue size:: ............................. 25
Maximum bond issue size:: ............................. 150
Issue multiple amount:: ............................... 10.0
Do It
```



```
Interest Rate Model Parameters for Market CANADA
Term of Long Rate: ...... 20.0
Term of Short Rate: .... 1.0
For yearly changes in bond rates:
  SD, Long Rate:: ....... 0.25
  SD, Short Rate:: ...... 0.5
  Alpha, Short Rate:: 0.5
For quarterly changes in bond rates:
  SD, Long Rate:: ....... 0.0625
  SD, Short Rate:: ...... 0.125
  Alpha, Short Rate:: 0.5
Do It
```

The process is repeated until no more markets are selected.

```
Do you need any new financial markets for your
analysis (Y/N)?
```

If Y, *a message is sent to add a new market. The process is repeated as many times as required.*

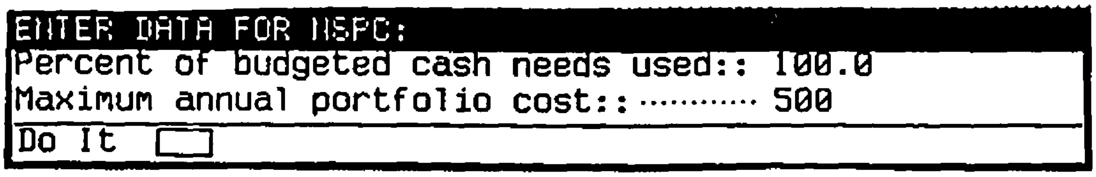
```
N
Is the problem's market knowledge complete and
correct (Y/N)?
```

If N, *the system returns to the beginning of the market-checking sequence.*

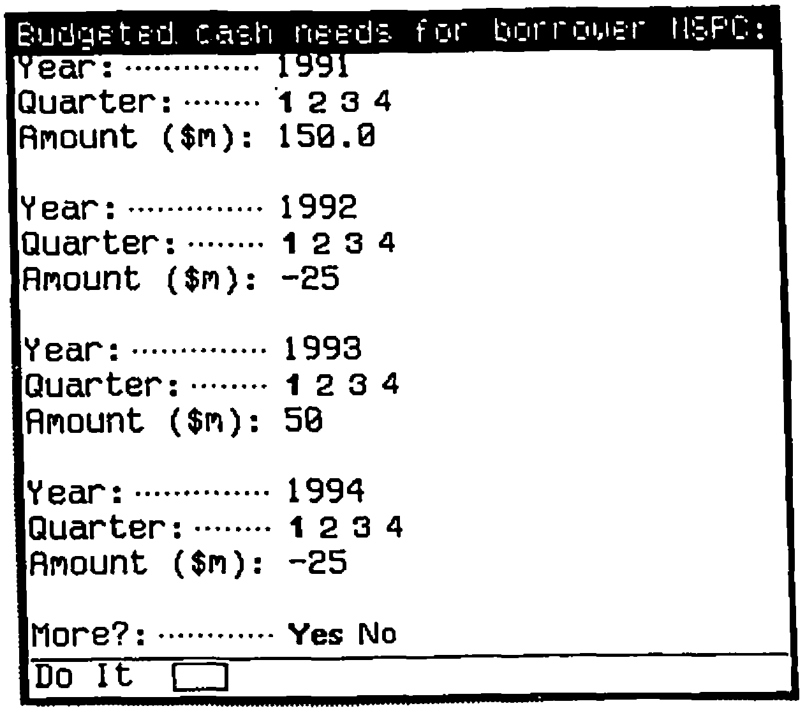
```
Y
```

2.2 Borrower requirements

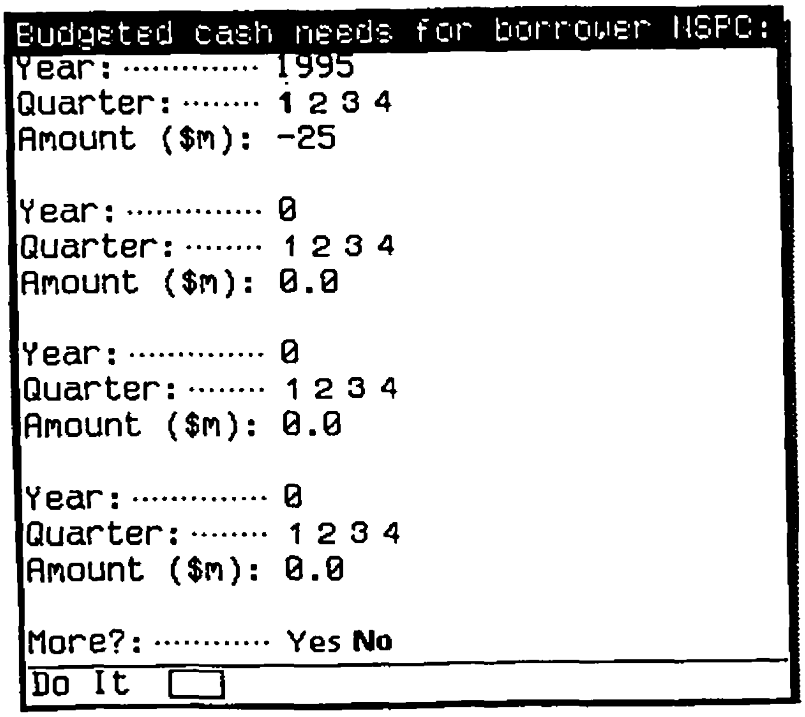
```
Next, you need to review the current borrower and
cash requirements.  OK (Y to confirm, S to skip,
select an alternate task)?

Y

The borrower NSPC is defined for this problem.
Please check that the parameters and cash
requirements for NSPC are correct.
```



```
ENTER DATA FOR NSPC:
Percent of budgeted cash needs used:: 100.0
Maximum annual portfolio cost::·········· 500
Do It
```



```
Budgeted cash needs for borrower NSPC:
Year: .............. 1991
Quarter: ........ [✓ 1] 2 3 4
Amount ($m): 150.0

Year: .............. 1992
Quarter: ........ [✓ 1] 2 3 4
Amount ($m): -25

Year: .............. 1993
Quarter: ........ [✓ 1] 2 3 4
Amount ($m): 50

Year: .............. 1994
Quarter: ........ [✓ 1] 2 3 4
Amount ($m): -25

More?: ............ [✓ Yes] No
Do It
```



```
Budgeted cash needs for borrower NSPC:
Year: .............. 1995
Quarter: ........ [✓ 1] 2 3 4
Amount ($m): -25

Year: .............. 0
Quarter: ........ 1 2 3 4
Amount ($m): 0.0

Year: .............. 0
Quarter: ........ 1 2 3 4
Amount ($m): 0.0

Year: .............. 0
Quarter: ........ 1 2 3 4
Amount ($m): 0.0

More?: ............ Yes [✓ No]
Do It
```

These forms change the borrower, if desired. If no borrower were defined in the problem space, one would be added instead.

```
Is the problem's borrower knowledge complete and correct?
```

Y

2.3 Future rates

```
Next, you should review the future rate scenarios
your analysis.  OK (Y to confirm, S to skip, N to
select an alternate task)?
```

If no rate scenarios were defined, the prompt would suggest adding them instead.

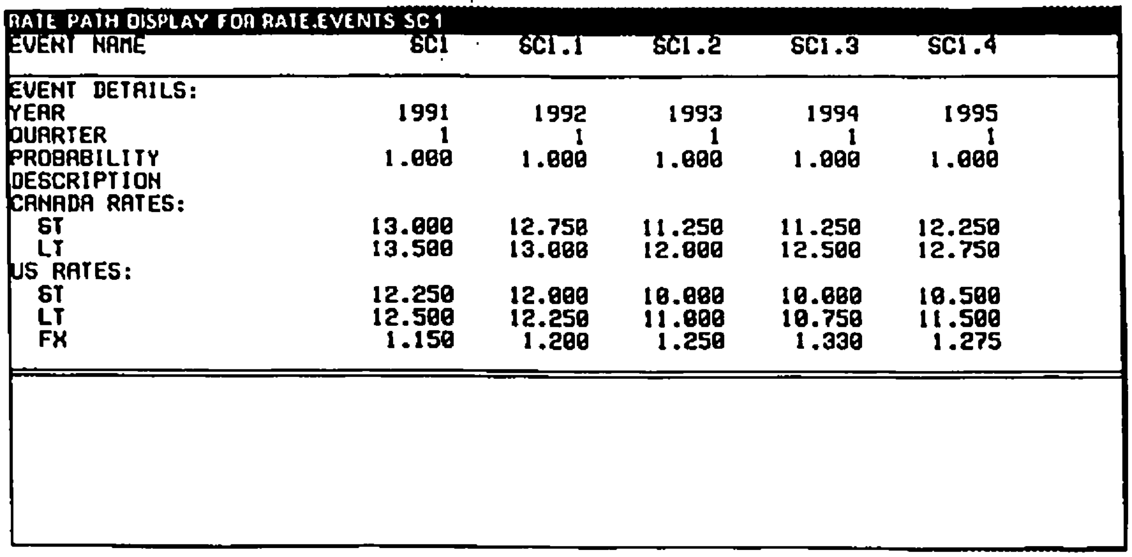
Y

```
Rate scenarios SC1 and SC2 are defined for this
problem.  Do you wish to review them (Y to confirm
S to skip, N to select an alternate task)?
```

Y

```
Choose a rate scenario to review (SC1/ SC2).
```

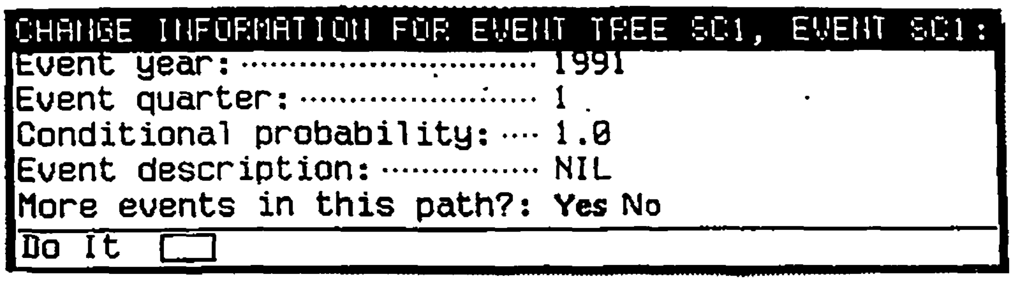
SC1



```
RATE PATH DISPLAY FOR RATE.EVENTS SC1
EVENT NAME              SC1      SC1.1     SC1.2     SC1.3     SC1.4

EVENT DETAILS:
YEAR                   1991       1992      1993      1994      1995
QUARTER                   1          1         1         1         1
PROBABILITY           1.000      1.000     1.000     1.000     1.000
DESCRIPTION
CANADA RATES:
  ST                 13.000     12.750    11.250    11.250    12.250
  LT                 13.500     13.000    12.000    12.500    12.750
US RATES:
  ST                 12.250     12.000    10.000    10.000    10.500
  LT                 12.500     12.250    11.000    10.750    11.500
  FX                  1.150      1.200     1.250     1.330     1.275
```

```
Do you wish to change SC1 (Y/N)?
```

Y



```
CHANGE INFORMATION FOR EVENT TREE SC1, EVENT SC1:
Event year: ........................... 1991
Event quarter: ........................ 1
Conditional probability: .... 1.0
Event description: ............... NIL
More events in this path?: Yes No
Do It
```

The form is repeated for each event in scenario SC1. The process is repeated until no more scenarios are selected.

```
Do you need any new rate scenarios for your
analysis (Y/N)?
```

If Y, *a message is sent to add a new rate scenario. The process is repeated as many times as required.*

N

```
Is the problem's future rate knowledge complete
and correct (Y/N)?
```

Y

3. Description of the current problem

3.1 Existing and new debts

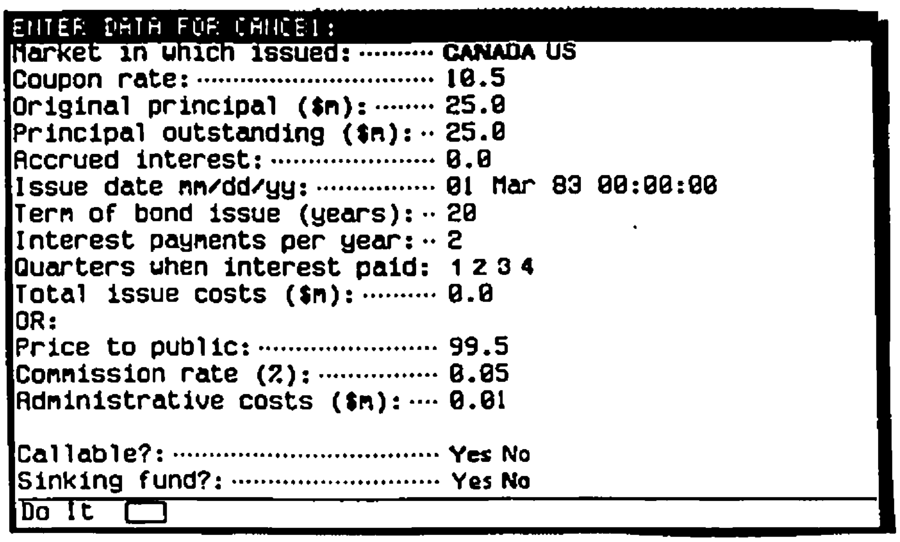
```
The debts now defined for the problem are CANBB1,
CANCB1, CANSFB1, CANCSFB1, USBB1, USCB1, USSFB1,
USCSFB1, CANBANK1 and CANPROV1.
Do you wish to review them (Y to confirm
```

S to skip, N to select an alternate task)?

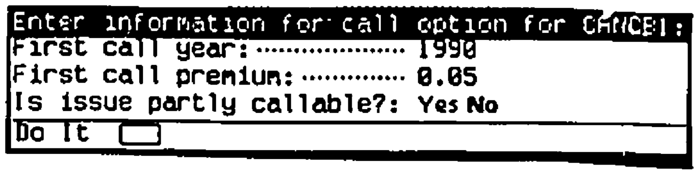
Y

Choose a debt to review (CANBB1, CANCB1, CANSFB1, USBB1, USSFB1 and CANBANK1).

CANCB1



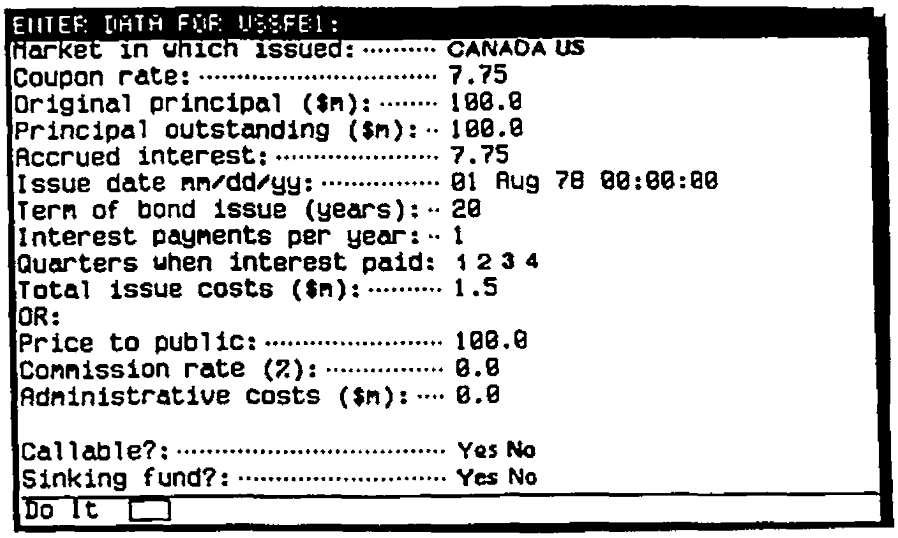
```
ENTER DATA FOR CANCB1:
Market in which issued:·········· [✓ CANADA] US
Coupon rate:····························· 10.5
Original principal ($m):········ 25.0
Principal outstanding ($m):·· 25.0
Accrued interest:················ 0.0
Issue date mm/dd/yy:·············· 01 Mar 83 00:00:00
Term of bond issue (years):·· 20
Interest payments per year:·· 2
Quarters when interest paid: 1 2 3 4
Total issue costs ($m):·········· 0.0
OR:
Price to public:·························· 99.5
Commission rate (%):················ 0.05
Administrative costs ($m):···· 0.01

Callable?:·································· [✓ Yes] No
Sinking fund?:···························· Yes [✓ No]
Do It
```



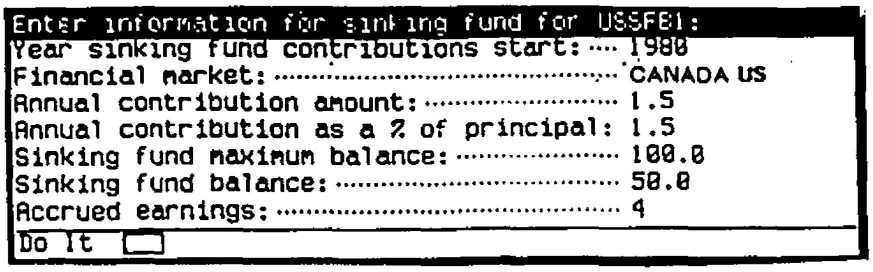
```
Enter information for call option for CANCB1:
First call year:····················· 1990
First call premium:·············· 0.05
Is issue partly callable?: Yes [✓ No]
Do It
```

Choose a debt to review (CANBB1, CANCB1, CANSFB1, USBB1, USSFB1 and CANBANK1.?

USSFB1



```
ENTER DATA FOR USSFB1:
Market in which issued: ·········· CANADA US
Coupon rate: ······················ 7.75
Original principal ($m): ·········· 100.0
Principal outstanding ($m): ······· 100.0
Accrued interest: ················· 7.75
Issue date mm/dd/yy: ·············· 01 Aug 78 00:00:00
Term of bond issue (years): ······· 20
Interest payments per year: ······· 1
Quarters when interest paid: 1 2 3 4
Total issue costs ($m): ··········· 1.5
OR:
Price to public: ·················· 100.0
Commission rate (%): ·············· 0.0
Administrative costs ($m): ········ 0.0

Callable?: ························ Yes No
Sinking fund?: ···················· Yes No
Do It
```



```
Enter information for sinking fund for USSFB1:
Year sinking fund contributions start: ···· 1980
Financial market: ························· CANADA US
Annual contribution amount: ··············· 1.5
Annual contribution as a % of principal: 1.5
Sinking fund maximum balance: ············· 100.0
Sinking fund balance: ····················· 50.0
Accrued earnings: ························· 4
Do It
```

The process is repeated until no more debts are selected.

Do you need any new debts for your analysis (Y/N)?

If Y, a message is sent to add a new debt. The process is repeated as many times as required.

N

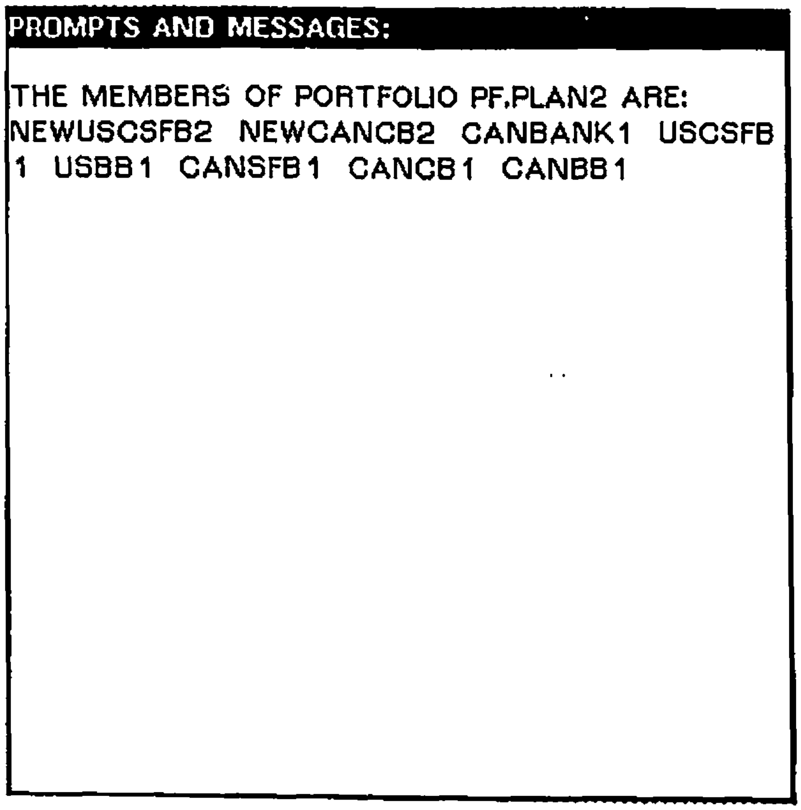
3.2 Debt portfolios

```
The debt portfolios now defined for the problem are
PF.PLAN1, PF.PLAN2.  Do you wish to review them
(Y to confirm, S to skip, N to select an alternate
task)?
```

Y

```
Choose a portfolio to review (PF.PLAN1, PF.PLAN2).
```

PF.PLAN2.



PROMPTS AND MESSAGES:

```
THE MEMBERS OF PORTFOLIO PF.PLAN2 ARE:
NEWUSCSFB2 NEWCANCB2 CANBANK1 USCSFB
1 USBB1 CANSFB1 CANCB1 CANBB1
```

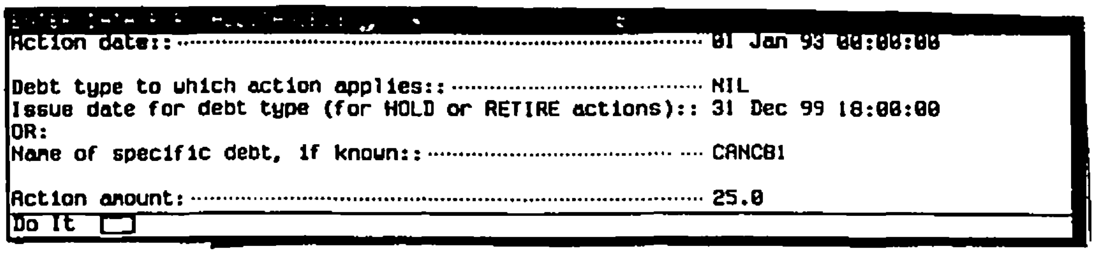
```
Do you wish to change PF.PLAN2 (Y/N)?

Y
```

A menu appears for re-selection of portfolio numbers. The process is repeated until no more portfolios are selected.

```
Do you need any new portfolios for your analysis
(Y/N)?

N

Are the problem's debt portfolios complete and
correct (Y/N)?

Y
```

3.3 Borrowing actions

```
For portfolio simulation and cash flow projection,
you need to specify the debt calls, short-term
borrowing and short-term debt repayments planned
during the planning period.  There are no borrowing
actions currently specified for this problem.  Do
you wish to create some borrowing actions (Y to
confirm, S to skip, N to select an alternate task)?

Y

Enter the name of a new borrowing action:

CALL.CANCB1
```



```
Action date::........................................ 01 Jan 93 00:00:00

Debt type to which action applies::.................. NIL
Issue date for debt type (for HOLD or RETIRE actions):: 31 Dec 99 18:00:00
OR:
Name of specific debt, if known::.................... CANC01

Action amount:....................................... 25.0
Do It
```

This form adds a borrowing action and is repeated as many times as necessary. If there were borrowing actions in the problem space, they would be reviewed prior to adding new ones. The process is repeated as many times as required.

```
Are the problem's borrowing actions complete and
correct (Y/N)?
```

Y

3.4 Final problem confirmation

```
The problem is now completely described.  Do you
wish to change anything before starting an analysis
(Y/N)?
```

If Y, *a list is displayed for selection of an item to be changed. This is repeated as many times as necessary.*

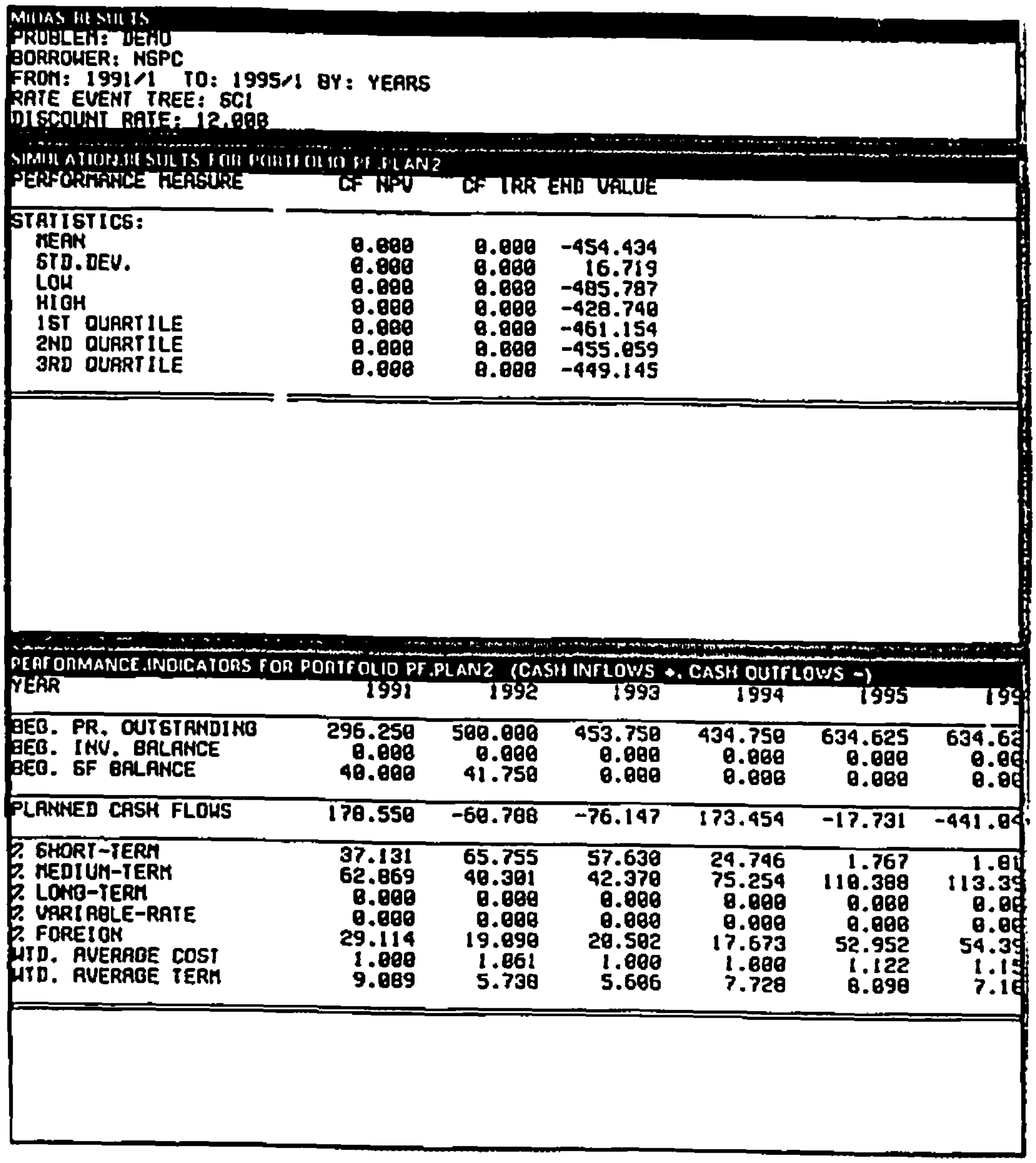
N

4. Portfolio simulation

```
You are now ready to do the analysis.  OK (Y/N)?
```

Y

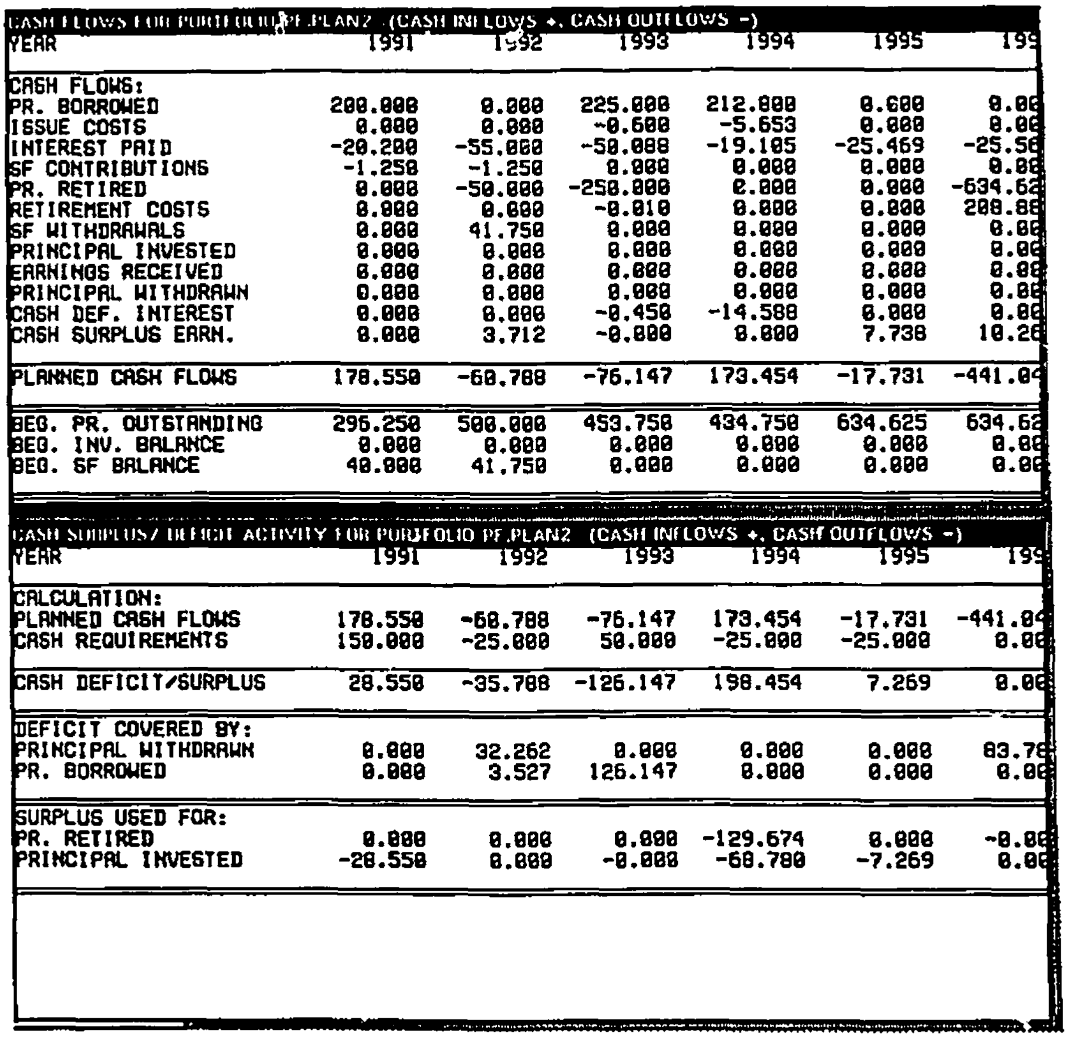
A message is sent to PORTFOLIOS: MTH.SIMULATE, *resulting in a menu for selection of the portfolio to be analyzed. During simulation, the pass count is displayed to indicate the progression of the simulation. On completion, the following output is displayed:*



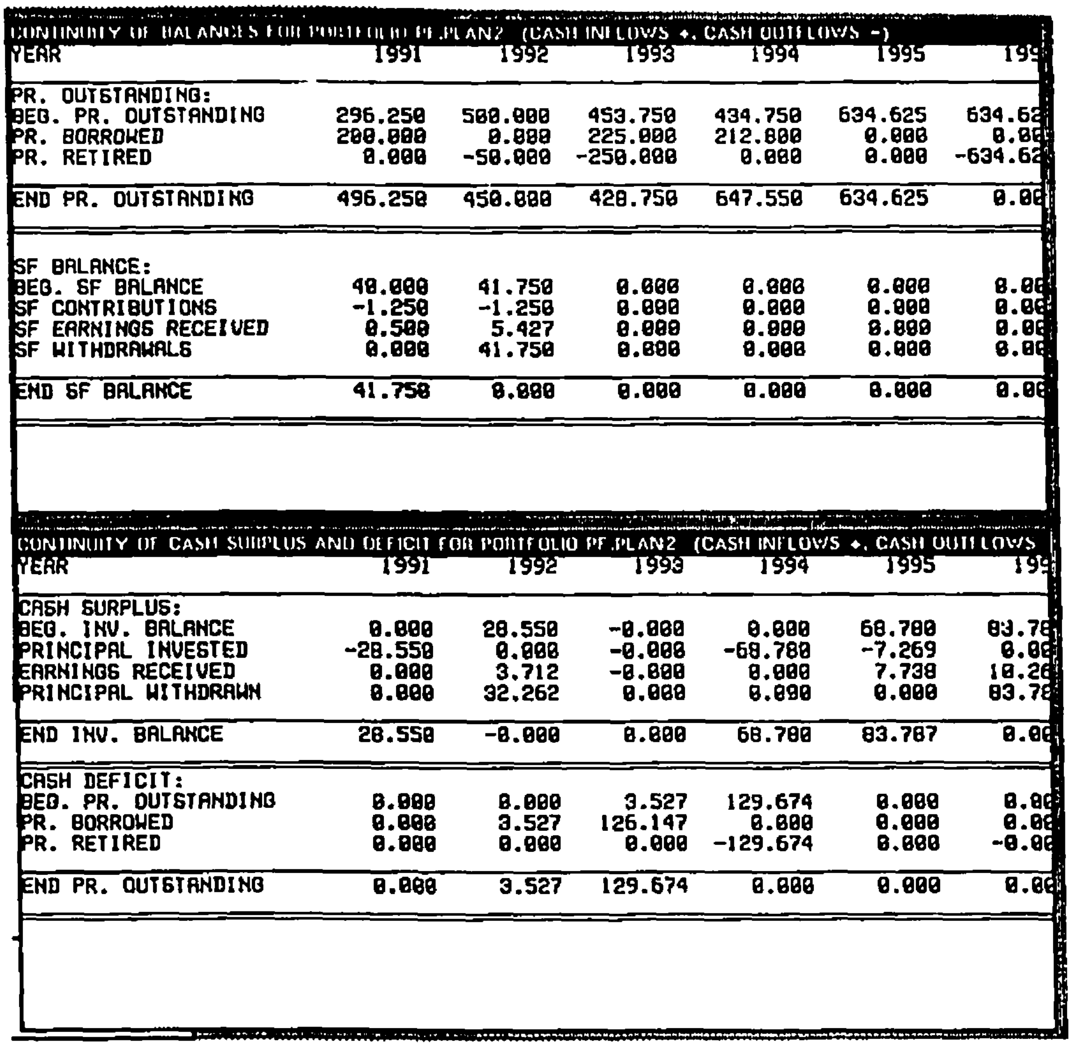
```
MIDAS RESULTS
PROBLEM: DEMO
BORROWER: NSPC
FROM: 1991/1  TO: 1995/1 BY: YEARS
RATE EVENT TREE: SC1
DISCOUNT RATE: 12.000

SIMULATION.RESULTS FOR PORTFOLIO PF.PLAN2
PERFORMANCE MEASURE            CF NPV    CF IRR END VALUE

STATISTICS:
  MEAN                          0.000     0.000  -454.434
  STD.DEV.                      0.000     0.000    16.719
  LOW                           0.000     0.000  -485.787
  HIGH                          0.000     0.000  -428.740
  1ST QUARTILE                  0.000     0.000  -461.154
  2ND QUARTILE                  0.000     0.000  -455.059
  3RD QUARTILE                  0.000     0.000  -449.145


PERFORMANCE.INDICATORS FOR PORTFOLIO PF.PLAN2  (CASH INFLOWS +, CASH OUTFLOWS -)
YEAR                          1991      1992      1993      1994      1995      19

BEG. PR. OUTSTANDING       296.250   500.000   453.750   434.750   634.625   634.62
BEG. INV. BALANCE            0.000     0.000     0.000     0.000     0.000     0.00
BEG. SF BALANCE             40.000    41.750     0.000     0.000     0.000     0.00

PLANNED CASH FLOWS         170.550   -60.708   -76.147   173.454   -17.731  -441.04

% SHORT-TERM                37.131    65.755    57.630    24.746     1.767     1.01
% MEDIUM-TERM               62.869    40.301    42.370    75.254   110.388   113.39
% LONG-TERM                  0.000     0.000     0.000     0.000     0.000     0.00
% VARIABLE-RATE              0.000     0.000     0.000     0.000     0.000     0.00
% FOREIGN                   29.114    19.090    20.502    17.673    52.952    54.39
WTD. AVERAGE COST            1.000     1.061     1.000     1.000     1.122     1.15
WTD. AVERAGE TERM            9.089     5.730     5.606     7.728     8.098     7.10
```

If you wish to see more details, use the DISPLAY by middle-clicking on the MIDAS icon. Enter Y to resume the assisted analysis process.

A number of detailed output displays are available through the display menu, including the following:



```
CASH FLOWS FOR PORTFOLIO PF.PLAN2  (CASH INFLOWS +, CASH OUTFLOWS -)
YEAR                       1991       1992       1993       1994       1995       199

CASH FLOWS:
PR. BORROWED            200.000      0.000    225.000    212.000      0.600      0.00
ISSUE COSTS               0.000      0.000     -0.600     -5.653      0.000      0.00
INTEREST PAID           -20.200    -55.000    -50.088    -19.105    -25.469    -25.50
SF CONTRIBUTIONS         -1.250     -1.250      0.000      0.000      0.000      0.00
PR. RETIRED               0.000    -50.000   -250.000      0.000      0.000   -634.62
RETIREMENT COSTS          0.000      0.000     -0.010      0.000      0.000    208.88
SF WITHDRAWALS            0.000     41.750      0.000      0.000      0.000      0.00
PRINCIPAL INVESTED        0.000      0.000      0.000      0.000      0.000      0.00
EARNINGS RECEIVED         0.000      0.000      0.000      0.000      0.000      0.00
PRINCIPAL WITHDRAWN       0.000      0.000      0.000      0.000      0.000      0.00
CASH DEF. INTEREST        0.000      0.000     -0.450    -14.588      0.000      0.00
CASH SURPLUS EARN.        0.000      3.712     -0.000      0.000      7.738     10.20

PLANNED CASH FLOWS      178.550    -60.788    -76.147    173.454    -17.731   -441.04

BEG. PR. OUTSTANDING    296.250    500.000    453.750    434.750    634.625    634.62
BEG. INV. BALANCE         0.000      0.000      0.000      0.000      0.000      0.00
BEG. SF BALANCE          40.000     41.750      0.000      0.000      0.000      0.00

CASH SURPLUS/ DEFICIT ACTIVITY FOR PORTFOLIO PF.PLAN2  (CASH INFLOWS +, CASH OUTFLOWS -)
YEAR                       1991       1992       1993       1994       1995       199

CALCULATION:
PLANNED CASH FLOWS      178.550    -60.788    -76.147    173.454    -17.731   -441.04
CASH REQUIREMENTS       150.000    -25.000     50.000    -25.000    -25.000      0.00

CASH DEFICIT/SURPLUS     28.550    -35.788   -126.147    198.454      7.269      0.00

DEFICIT COVERED BY:
PRINCIPAL WITHDRAWN       0.000     32.262      0.000      0.000      0.000     83.70
PR. BORROWED              0.000      3.527    126.147      0.000      0.000      0.00

SURPLUS USED FOR:
PR. RETIRED               0.000      0.000      0.000   -129.674      0.000     -0.00
PRINCIPAL INVESTED      -28.550      0.000     -0.000    -68.780     -7.269      0.00
```

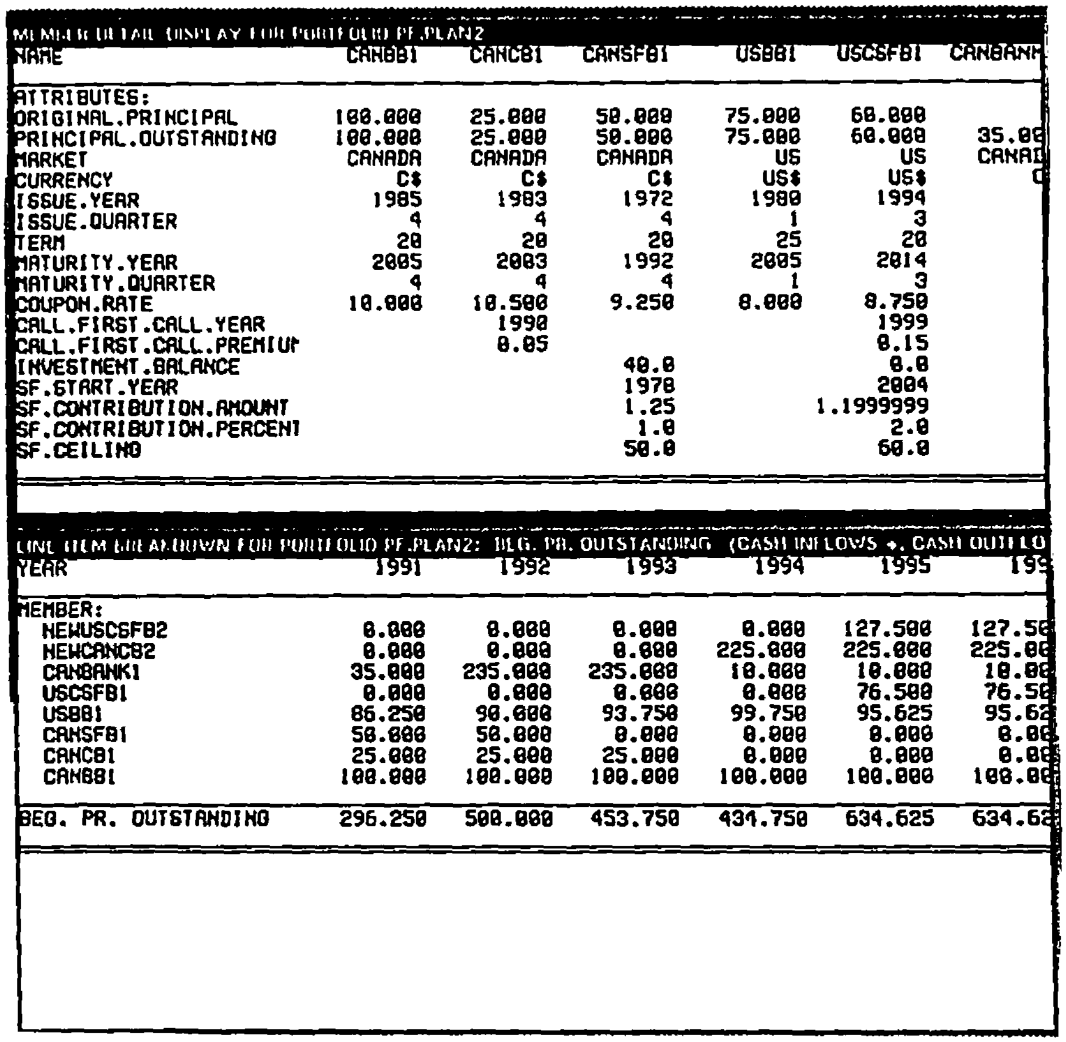


CONTINUITY OF BALANCES FOR PORTFOLIO PF.PLAN2 (CASH INFLOWS +, CASH OUTFLOWS -)

YEAR	1991	1992	1993	1994	1995	199
PR. OUTSTANDING:						
BEG. PR. OUTSTANDING	296.250	500.000	453.750	434.750	634.625	634.62
PR. BORROWED	200.000	0.000	225.000	212.800	0.000	0.00
PR. RETIRED	0.000	-50.000	-250.000	0.000	0.000	-634.62
END PR. OUTSTANDING	496.250	450.000	428.750	647.550	634.625	0.00
SF BALANCE:						
BEG. SF BALANCE	40.000	41.750	0.000	0.000	0.000	0.00
SF CONTRIBUTIONS	-1.250	-1.250	0.000	0.000	0.000	0.00
SF EARNINGS RECEIVED	0.500	5.427	0.000	0.000	0.000	0.00
SF WITHDRAWALS	0.000	41.750	0.000	0.000	0.000	0.00
END SF BALANCE	41.750	0.000	0.000	0.000	0.000	0.00

CONTINUITY OF CASH SURPLUS AND DEFICIT FOR PORTFOLIO PF.PLAN2 (CASH INFLOWS +, CASH OUTFLOWS

YEAR	1991	1992	1993	1994	1995	199
CASH SURPLUS:						
BEG. INV. BALANCE	0.000	28.550	-0.000	0.000	68.700	83.78
PRINCIPAL INVESTED	-28.550	0.000	-0.000	-68.780	-7.269	0.00
EARNINGS RECEIVED	0.000	3.712	-0.000	0.000	7.738	10.26
PRINCIPAL WITHDRAWN	0.000	32.262	0.000	0.090	0.000	83.78
END INV. BALANCE	28.550	-0.000	0.000	68.700	83.787	0.00
CASH DEFICIT:						
BEG. PR. OUTSTANDING	0.000	0.000	3.527	129.674	0.000	0.00
PR. BORROWED	0.000	3.527	126.147	0.000	0.000	0.00
PR. RETIRED	0.000	0.000	0.000	-129.674	0.000	-0.00
END PR. OUTSTANDING	0.000	3.527	129.674	0.000	0.000	0.00



```
MEMBER DETAIL DISPLAY FOR PORTFOLIO PF.PLAN2
NAME                           CANBB1     CANCB1     CANSFB1       USBB1     USCSFB1    CANBANK

ATTRIBUTES:
ORIGINAL.PRINCIPAL            100.000     25.000     50.000       75.000     60.000
PRINCIPAL.OUTSTANDING         100.000     25.000     50.000       75.000     60.000      35.00
MARKET                         CANADA     CANADA     CANADA           US         US      CANAD
CURRENCY                           C$         C$         C$          US$        US$          C
ISSUE.YEAR                       1985       1983       1972         1980       1994
ISSUE.QUARTER                       4          4          4            1          3
TERM                               20         20         20           25         20
MATURITY.YEAR                    2005       2003       1992         2005       2014
MATURITY.QUARTER                    4          4          4            1          3
COUPON.RATE                    10.000     10.500      9.250        8.000      8.750
CALL.FIRST.CALL.YEAR                        1990                              1999
CALL.FIRST.CALL.PREMIUM                     0.05                              0.15
INVESTMENT.BALANCE                                     40.0                    0.0
SF.START.YEAR                                          1978                   2004
SF.CONTRIBUTION.AMOUNT                                 1.25              1.1999999
SF.CONTRIBUTION.PERCENT                                 1.0                    2.0
SF.CEILING                                             50.0                   60.0

LINE ITEM BREAKDOWN FOR PORTFOLIO PF.PLAN2: BEG. PR. OUTSTANDING  (CASH INFLOWS +. CASH OUTFLO
YEAR                             1991       1992       1993        1994       1995        199

MEMBER:
  NEWUSCSFB2                    0.000      0.000      0.000       0.000    127.500     127.50
  NEWCANCB2                     0.000      0.000      0.000     225.000    225.000     225.00
  CANBANK1                     35.000    235.000    235.000      10.000     10.000      10.00
  USCSFB1                       0.000      0.000      0.000       0.000     76.500      76.50
  USBB1                        86.250     90.000     93.750      99.750     95.625      95.62
  CANSFB1                      50.000     50.000      0.000       0.000      0.000       0.00
  CANCB1                       25.000     25.000     25.000       0.000      0.000       0.00
  CANBB1                      100.000    100.000    100.000     100.000    100.000     100.00

BEG. PR. OUTSTANDING          296.250    500.000    453.750     434.750    634.625     634.62
```

```
Y

Do you wish to do other analyses now (Y/N)?

N

Do you wish to save your problem (Y/N)?

Y

Problem space DEMO successfully saved.

This is the end of the planning session.
```

11
Findings and Conclusions

The MIDAS project has produced a number of observations and conclusions about various aspects of the broad, multidisciplinary problem under investigation. This chapter closes the dissertation, summarizing the project's contributions and suggests several areas for extensions of this work, both to broaden the capabilities of systems such as MIDAS and to study related decision support issues.

1. Research contributions

As outlined in the introduction to this dissertation, the broad purpose of this research has been to demonstrate the integration of multiple modelling and expert systems techniques to extend the capabilities of a domain-specific decision support system beyond the range available with commonly-used financial DSS tools. Specifically, the project addressed the following issues:

1. The feasibility of a hierarchical planning approach applying stochastic programming, simulation modelling and cash flow projection in a complementary manner to corporate debt planning.
2. The feasibility of integrated frame- and rule-based modelling and system support, including task selection; model formulation, solution and modification; and output data management.
3. The feasibility of integrating multiple model types and heuristic reasoning through a common, object-oriented domain representation.
4. A design approach for user modelling assistance, including parametric and key factor impact analysis and explanation of the results of cash flow projections and financial simulations.

5. Identification of design principles for dynamic stochastic portfolio management decision support systems for users who are not modelling experts.

Overall, this project has demonstrated the feasibility of knowledge-based decision support for debt management. The object-oriented design approach functions well for the selected domain and should extend well to other problems exhibiting such a modular problem structure. The prototype system developed as part of this project requires only the addition of limited, clearly defined error-checking features to become a fully-functional system usable in a corporate setting.

Consideration of the specific research questions has led to a number of findings and conclusions, as follows:

1. A hierarchical planning approach applying stochastic programming, simulation modelling and cash flow projection to corporate debt planning underlies the design of the entire MIDAS system and its technical feasibility has been demonstrated in the prototype. While its practical utility remains to be empirically verified through longer-term corporate use, initial reaction from corporate and financial professionals is extremely promising.

2. Proposed designs for integrated frame- and rule-based modelling and system support have been presented in Chapters 6 and 7, and demonstrated in the prototype system.

3. As discussed in Chapters 3 and 6, the domain model provides a natural organization for modelling components and knowledge, formulating and integrating multiple complex model types and maintaining their consistency in a manner which mimics the operation of the physical system being modelled. This approach can also be viewed as one of manipulating the single underlying domain model with multiple operators or solution techniques, including optimization, simulation and heuristic reasoning. This view presents some practical and conceptual difficulties because definitions of 'model' vary among researchers and because solution of simulation models is difficult to separate from their specifi-

cation (Dolk 1990); to the extent that it does describe MIDAS' approach to an integrated modelling environment, it indicates the feasibility of the separation of models and operators in a practical setting.

An additional benefit of the underlying domain representation is that it provides a clear semantic starting point for communication between system users and developers, eliminating some of the 'language barriers' that face these two groups during the development of a complex system such as MIDAS.

The domain representation also provides the problem description used by rules for refining initial borrowing plans, assisting users in choosing task sequences, reducing optimization problem sizes, analyzing results and maintaining knowledge base consistency. These integrated heuristics add more 'human' expertise to the modelling environment than would be practical with algorithmic procedures alone.

4. A design approach for user assistance has been presented in Chapter 8. This feature, together with the task management component of the system support subsystem, adds human intermediary behaviour to the system by incorporating abstract tasks and modelling, manipulated by generalized reasoning or control processes. This approach allows extension of system capabilities declaratively through the addition of more knowledge frames and it appears readily applicable to additional intermediary functions.

5. Design principles for dynamic stochastic portfolio management decision support systems were identified in Chapter 4. They include:
   a. Frame-based knowledge representation
   b. Object-oriented modelling
   c. Spreadsheet-oriented financial model structure
   d. Separation of knowledge and control/reasoning
   e. Model and rule integration through an underlying domain representation.

These design principles reflect an integrated object- , procedure- and rule-based approach to portfolio management DSS design and construction, both for complex modelling and for user assistance in modelling. The utility of this approach has been demonstrated in the design and prototype, where it is seen to apply to a variety of functions requiring different levels of abstraction and reasoning. For the complex situations analyzed by a debt management decision support system, such a combination of tools has been shown to provide more flexibility and power than does any one tool individually, just as a human analyst must employ a variety of skills.

These design principles and, indeed, the MIDAS architecture, should extend directly to planning and portfolio management problems in a wide range of additional domains. Examples include financial institution asset/liability management, personal or corporate investment portfolio management, corporate management of capital project portfolios, research and development planning and configuration of waste management facilities to meet environmental constraints. All these domains require construction of portfolios of domain objects (investments, debts, projects, equipment components, processes) to meet future requirements under uncertainty; while they vary in the description of specific domain objects and in the degree of interaction among portfolio members, the essential analytical techniques and system design are likely to be applicable with only minor changes.

2. Possible system improvements and extensions

MIDAS requires implementation of the designed LP support and user assistance components in order to fully support hierarchical debt planning. Other extensions which would make it more functional in a practical situation are:

1. Rule- and/or method-based data entry edits and consistency checks to ensure the continuing quality of the underlying domain representation.
2. Improvement of the system's pseudo-random rate generation models based on financial analyst expertise. This would provide more realism in the rate gen-

eration models, including yield curves which better reflect actual curve shapes, realistic model parameters, more realistic rate spreads, some linking of interest and foreign exchange rates, and possibly the inclusion of rollover risk through rate variations based on future changes in the Corporation's credit rating.

3. A more flexible interface for rate event tree input which would allow users to graphically display and manipulate a tree during problem specification.

4. Extension of the cash flow projection facility to produce projected financial statement results according to generally accepted accounting principles. These would include discount and premium amortization, amortization of foreign exchange gains and losses, allocaton of interest costs to time periods independent of payment dates, and other adjustments which determine the Corporation financial results on which rate levels are based.

5. Inclusion of all debt types now used by the Corporation, as listed in Chapter 4.

6. Inclusion of the ability to create new debt types as combinations of components, as outlined in Chapter 6.

7. Inclusion of a mechanism to diagnose and handle infeasibility conditions for the optimization model. This may come about through work with the ANALYZE system.

8. Design and implementation of the other requirements listed in Chapter 4.

The overall modular design of the system would appear to allow such extensions without changing its basic architecture.

In the longer run, modifications of the system to run in a more fully distributed environment, taking advantage of software outside KEE for simulation and output management, is a possible area of further work, as will application of the system's design approach to other domains including bank asset/liability management and corporate financial planning.

3. Suggested further research

This project has suggested a number of areas for further research, as follows:

1. The nature of problems for which the object-oriented DSS design approach is appropriate. Can problem dimensions or characteristics be identified which are associated with probable success of an object-oriented DSS in the way that 'good' expert systems projects have come to be identified?

2. The feasibility of and design for a user-controlled, object-oriented financial modelling shell. The modelling subsystem designed and prototyped for this project is apparently the first published application of object-oriented programming to large-scale financial modelling. This approach would appear to offer significant additional power if enhanced by a graphic, spreadsheet or equation-oriented user modelling interface, since reusable model objects could then be easily created and maintained by users. The resulting integrated modelling environment could greatly simplify the direct construction and manipulation of financial models by end-users.

3. User needs for complex model explanation. Empirical explanation research to date has not considered complex financial models. What types of explanations of these models do various users require? What knowledge is required for these explanations, and what reasoning processes are used by experts to build the explanations?

4. The feasibility of integrating into systems such as MIDAS the existing linear programming model explanation capabilities of systems such as ANALYZE (Greenberg 1987*a*,*b*, 1988, 1989; Greenberg and Lundgren 1989; Greenberg and Murphy 1989).

5. Identification of an improved design approach for explanation of complex financial simulations and projections, in particular by developing self-explaining models. Is it possible to build a system for complex financial simulation and

projection which integrates quantitive and qualitative modelling knowledge and so generates human-like explanations without a complete duplicate qualitative model representation?

6. Improved techniques for controlling and documenting the interdependence of individual methods within complex models such as those in MIDAS.

7. Improved documentation techniques for object-oriented analysis and design, preferably building on the structured analysis and design techniques already in common corporate use.

8. Empirical investigation of the degree to which 'intelligence' in DSS's promotes the use by managers of complex models over more simple ones. Chapter 2 outlined extensive research identifying barriers to corporate use of complex financial models. As 'smarter' DSS's are developed and put into use, follow-up research to determine their effectiveness will help to further identify the real factors which inhibit or promote their use for corporate decision support.

In conclusion, this project has integrated two techniques—management science modelling and artificial intelligence—to provide a decision support tool with greater capabilities than currently exists for financial planning. It has also demonstrated a hierarchical planning approach to debt planning which overcomes the limitations of using any single modelling or heuristic approach. In so doing, it moves the capabilities of financial decision support systems closer to the ideal in which models can be readily used and understood by non-experts without human intervention.

Appendix A
MIDAS Rate Model Specification

Notation and comments

s,t= $0, \ldots, T+1$ denote time periods.

0 indicates time immediately prior to the start of the planning period.

T is the length of the planning period.

T+1 indicates time immediately after the end of the planning period.

$m = 1, \ldots, M$ denotes a financial market.

$k = 1, \ldots, K$ denotes an available debt type, distinguished by market, term, debt class (bond, short-term credit etc.) and features (call option, sinking fund etc.)

$j = 1, \ldots, J$ denotes a future (rate) scenario

$e_j = e_{j1}, \ldots, e_{jT}$ denotes a sequence of (rate) events specifying mean interest and exchange rates in time periods t for scenario $j = 1, \ldots, J$

(e_j) indicates that a variable or parameter is contingent on the event sequence e_j

All rates are annual unless otherwise noted.

1. Random number generation

1.1 Input parameter

$x^i_{[0,1)}$ (for $i = 1, \ldots, n$) realization of a random variable with uniform distribution between 0 (inclusive) and 1 (exclusive)

1.2 Output

$z_{(0,1)}$ realization of a random variable with a normal distribution with mean 0 and standard deviation 1

$$z_{(0,1)} = \sqrt{-2 \ln x^1_{[0,1)}} \cos(2\pi x^2_{[0,1)})$$

2. Interest rate model

For $j = 1,\ldots,J$, $s = 0,\ldots,T$, $t = 1,\ldots,T+1$, $m = 1,\ldots,M$ and $k = 1,\ldots,K$, where k denotes a type of debt:

2.1 Input parameters

τl^m	term in years of 'long-term' debt in market m
τs^m	term in years of 'short-term' debt in market m
$\bar{r}l_t^m(e_j)$	mean long-term government interest rate in market m at the beginning of time period t
$\bar{r}s_t^m(e_j)$	mean short-term government interest rate in market m at the beginning of time period t
sp^k	mean spread between the market rate for debt type k and the rate for government debt of the same remaining term (time outstanding)
spy^k	mean spread between the market rate for debt type k, assuming k is called at its first call date, and the rate for government debt of the same remaining term, where k is a callable bond
$spsf^m$	mean spread between the government short-term interest rate and the sinking fund earnings rate in market m
σal^m	standard deviation of the annual change in the government long-term rate in market m
σql^m	standard deviation of the quarterly change in the government long-term rate in market m
αa^m	proportion of the annual change in the government long-term rate included in the annual change in the government short-term rate in market m
αq^m	proportion of the quarterly change in the government long-term rate included in the quarterly change in the government short-term rate in market m
σas^m	standard deviation of the random component of the annual change in the government short-term rate in market m
σqs^m	standard deviation of the random component of the quarterly change in the government short-term rate in market m

$\tau^k_{s,t}$ remaining term in years at the beginning of time period t for debt type k issued in period s

$\tau y^k_{s,t}$ * remaining time in years from the beginning of period t to the first call date for debt type k issued in period s, where k is a callable bond

$rl^m_0 = \bar{r}l^m_0 = \tilde{r}l^m_0$ long-term government interest rate in market m immediately prior to time period 1

$rs^m_0 = \bar{r}s^m_0 = \tilde{r}s^m_0$ short-term government interest rate in market m immediately prior to time period 1.

2.2 Output

$\Delta\bar{r}l^m_t(e_j)$ change in the mean long-term government interest rate in market m between time period $t-1$ and time period t

$$\Delta\bar{r}l^m_t(e_j) = \bar{r}l^m_t(e_j) - \bar{r}l^m_{t-1}(e_j)$$

$\Delta\bar{r}s^m_t(e_j)$ change in the mean short-term government interest rate in market m between time period $t-1$ and time period t

$$\Delta\bar{r}s^m_t(e_j) = \bar{r}s^m_t(e_j) - \bar{r}s^m_{t-1}(e_j)$$

$\Delta\tilde{r}pl^m_t(e_j)$ random part of the change in the long-term government interest rate in market m between time period $t-1$ and time period t

$$\Delta\tilde{r}pl^m_t(e_j) = \begin{cases} \sigma al^m z_{0,1} \\ \quad \text{for annual time periods } t \\ \sigma ql^m z_{0,1} \\ \quad \text{for quarterly time periods } t \end{cases}$$

$\tilde{r}l^m_t(e_j)$ random long-term government interest rate in market m at the beginning of time period t

$$\tilde{r}l^m_t(e_j) = \tilde{r}l^m_{t-1}(e_j) + \Delta\tilde{r}pl^m_t(e_j) + \Delta\bar{r}l^m_t(e_j)$$
$$\text{for annual time periods } t$$

$\Delta\tilde{r}ps^m_t(e_j)$ random part of the change in the short-term government interest rate in market m between time period $t-1$ and time period t

$$\Delta\tilde{r}ps^m_t(e_j) = \begin{cases} \sigma as^m z_{0,1} \\ \quad \text{for annual time periods } t \\ \sigma qs^m z_{0,1} \\ \quad \text{for quarterly time periods } t \end{cases}$$

$\widetilde{rs}_t^m(e_j)$ random snort-term government interest rate in market m at the beginning of time period t

$$\widetilde{rs}_t^m(e_j) = \begin{cases} \widetilde{rs}_{t-1}^m(e_j) + \Delta\widetilde{rs}_t^m(e_j) + \alpha a^m \Delta\widetilde{rpl}_t^m(e_j) + \Delta\widetilde{rps}_t^m(e_j) \\ \quad \text{for annual time periods } t \\ \widetilde{rs}_{t-1}^m(e_j) + \Delta\widetilde{rs}_t^m(e_j) + \alpha q^m \Delta\widetilde{rpl}_t^m(e_j) + \Delta\widetilde{rps}_t^m(e_j) \\ \quad \text{for quarterly time periods } t \end{cases}$$

$\bar{a}_t^m(e_j)$ a-coefficient of the yield curve for government securities in market m at the beginning of time period t, based on mean rates

$$\bar{a}_t^m(e_j) = \frac{\bar{rs}_t^m(e_j)\ln(\tau l^m) - \bar{rl}_t^m(e_j)\ln(\tau s^m)}{\ln(\tau l^m) - \ln(\tau s^m)}$$

$\bar{b}_t^m(e_j)$ b-coefficient of the yield curve for government securities in market m at the beginning of time period t, based on mean rates

$$\bar{b}_t^m(e_j) = \frac{\bar{rs}_t^m(e_j) - \bar{rl}_t^m(e_j)}{\ln(\tau s^m) - \ln(\tau l^m)}$$

$\tilde{a}_t^m(e_j)$ a-coefficient of the yield curve for government securities in market m, at the beginning of time period t, based on randomly generated rates

$$\tilde{a}_t^m(e_j) = \frac{\widetilde{rs}_t^m(e_j)\ln(\tau l^m) - \widetilde{rl}_t^m(e_j)\ln(\tau s^m)}{\ln(\tau l^m) - \ln(\tau s^m)}$$

$\tilde{b}_t^m(e_j)$ b-coefficient of the yield curve for government securities in market m, at the beginning of time period t, based on randomly generated rates

$$\tilde{b}_t^m(e_j) = \frac{\widetilde{rs}_t^m(e_j) - \widetilde{rl}_t^m(e_j)}{\ln(\tau s^m) - \ln(\tau l^m)}$$

$r_{s,t}^k(e_j)$ market interest rate at the beginning of period t for debt type k issued in period s

$$r_{s,t}^k(e_j) = \begin{cases} \bar{a}_t^m(e_j) + \bar{b}_t^m(e_j)\ln(\tau_{s,t}^k) + sp^k & \text{if based on mean yield curve} \\ \tilde{a}_t^m(e_j) + \tilde{b}_t^m(e_j)\ln(\tau_{s,t}^k) + sp^k & \text{if based on randomly generated yield curve} \end{cases}$$

for k not a callable bond and m is the market in which debt type k is issued

$$r_{s,t}^k(e_j) = \begin{cases} \bar{a}_t^m(e_j) + \bar{b}_t^m(e_j)\ln(r_{s,t}^k) + spy^k & \text{if based on mean yield curve} \\ \tilde{a}_t^m(e_j) + \tilde{b}_t^m(e_j)\ln(r_{s,t}^k) + spy^k & \text{if based on randomly generated yield curve} \end{cases}$$

for y a callable bond and m is the market in which debt type k is issued

$\bar{s}fe_t^m(e_j)$ earnings rate in period t for sinking funds in market m, based on mean rates

$$\bar{s}fe_t^m(e_j) = \bar{r}s_t^m(e_j) + spsf^m.$$

$\tilde{s}fe_t^m(e_j)$ earnings rate in period t for sinking funds in market m, based on randomly-generated rates

$$\tilde{s}fe_t^m(e_j) = \tilde{r}s_t^m(e_j) + spsf^m.$$

---

3. Foreign exchange rate model

For $j = 1, \ldots, J$, $t = 1, \ldots, T+1$, $m = 1, \ldots, M$ and $k = 1, \ldots, K$:

3.1 Input parameters

$\bar{p}_t^m(e_j)$ mean foreign exchange rate with respect to the Canadian dollar for market m at the beginning of time period t

σap^m standard deviation of the annual foreign exchange rate change for market m

σqp^m standard deviation of the quarterly foreign exchange rate change for market m

$p_0^m = \bar{p}_0^m = \tilde{p}_0^m$ foreign exchange rate in market m immediately prior to time period 1

3.2 Output

$\Delta\bar{p}_t^m(e_j)$ mean change in the foreign exchange rate for market m between time period $t-1$ and time period t

$$\Delta\bar{p}_t^m(e_j) = \bar{p}_t^m(e_j) - \bar{p}_{t-1}^m(e_j)$$

$\tilde{p}_t^m(e_j)$ randomly-generated foreign exchange rate for market m

at the beginning of time period t

$$\tilde{\rho}_t^m(e_j) = \begin{cases} \tilde{\rho}_{t-1}^m(e_j) + \sigma a \rho^m z_{0,1} + \Delta\bar{\rho}_t^m(e_j) \\ \quad \text{for annual time periods t} \\ \tilde{\rho}_{t-1}^m(e_j) + \sigma q \rho^m z_{0,1} + \Delta\bar{\rho}_t^m(e_j) \\ \quad \text{for quarterly time periods t} \end{cases}$$

$\rho_t^m(e_j)$ foreign exchange rate in period t for debt type k

$$\rho = \begin{cases} \bar{\rho}_t^m(e_j) & \text{if based on mean rates} \\ \tilde{\rho}_t^m(e_j) & \text{if based on randomly generated rates} \end{cases}$$

where m is the market in which debt type k is issued

4. Cash surplus and deficit rates

For $j = 1,\ldots,J$, $t = 1,\ldots,T+1$, $m = 1,\ldots,M$ and $k = 1,\ldots,K$:

4.1 Input parameters

spi^m mean spread between the short-term government securities rate and the short-term deposit earnings rate in market m

spd^m mean spread between the short-term government securities rate and the interest rate for short-term penalty borrowing in market m

4.2 Output

$i_t^m(e_j)$ short-term deposit earnings rate in market m for period t

$$i_t^m(e_j) = \begin{cases} \bar{rs}_t^m(e_j) + spi^m & \text{if based on mean rates} \\ \tilde{rs}_t^m(e_j) + spi^m & \text{if based on randomly generated rates} \end{cases}$$

rd_t^m short-term interest rate for penalty borrowing in market m

$$rd_t^m(e_j) = \begin{cases} \bar{rs}_t^m(e_j) + spd^m & \text{if based on mean rates} \\ \tilde{rs}_t^m(e_j) + spd^m & \text{if based on randomly generated rates} \end{cases}$$

Appendix B

MIDAS Simulation Model Specification

Notation and comments

$s,t = 0, \ldots, T+1$ denote time periods.

0 indicates the time period immediately prior to the start of the planning period.

T is the planning horizon or the length of the planning period.

$T+1$ indicates time immediately after the end of the planning period.

$m = 1, \ldots, M$ denotes a financial market.

$k = 1, \ldots, K$ denotes an available debt type, distinguished by market, term, debt class (bond, short-term credit etc.) and features (call option, sinking fund etc.). This specification covers bonds, call options, sinking funds and short-term credit in foreign and domestic markets.

$e_j = e_{j1}, \ldots, e_{jT}$ denotes a sequence of (rate) events specifying mean interest and exchange rates in time periods t for scenario $j = 1, \ldots, J$

(e_j) indicates that a variable or parameter is contingent on the event sequence e_j.

Interest, earnings and constraint satisfaction indicator calculations are given for yearly time period t: amounts are prorated for quarterly time periods.

All cash surplus investment and penalty borrowing is done in domestic currency.

1. Single-debt model

For $j = 1, \ldots, J$, $s = 0, \ldots, t-1$, $t = 1, \ldots, T$, $m = 1, \ldots, M$, and $k = 1, \ldots, K$, unless otherwise noted:

1.1 Input parameters

$p(e_j)$	probability of the event sequence e_j.
$B_t^k(e_j)$	dollar amount at par of debt type k issued at the beginning of period t.
$R_{s,t}^k(e_j)$	dollar amount at par of debt type k issued at the beginning of period s and retired at the beginning of period t.
$O_{0,1}^k(e_j)$	dollar amount at par of debt type k issued prior to the start of the planning period and outstanding at the beginning of period 1.
$SF_{0,1}^k$	dollar balance of the sinking fund for debt type k issued prior to the start of the planning period and outstanding at the beginning of period 1.
$r_{0,0}^k(e_j)$	interest rate in period t for debt type k outstanding at the beginning of period 1, where k denotes a type of bond.
$IA_{0,1}^k(e_j)$	accrued interest at the beginning of period 1 for debt type k outstanding at the start of the planning period.
f^k	issue cost per dollar borrowed for debt type k.
$g_{s,t}^k(e_j)$	retirement cost per dollar for debt type k issued in period s and retired at the beginning of period t.
$sfc_{s,t}^k$	sinking fund contribution in period t per dollar of principal outstanding, for debt type k issued in period s.
d	borrower discount rate to be used for net present value calculations.
$r_{s,t}^k(e_j)$	mean or randomly-generated market interest rate in period t for debt type k issued in period s.
$rsf_{s,t}^k(e_j)$	mean or randomly-generated sinking fund earnings rate in period t for debt type k issued in period s.
$\tau_{s,t}^k$	remaining term in years at the beginning of period t for debt type k issued in period s.

$\tau y^k_{s,t}(e_j)$	remaining time in years from the beginning of period t to the first call date for debt type k issued in period s, where k denotes a callable bond.
ρ^k_t	mean or randomly-generated foreign exchange rate in period t for debt type k.

1.2 Output

1.2.1 Cash flows and balances for a single event sequence (e_j)

In currency of issue:

$F^k_t(e_j)$ issue costs for debt type k issued in period t

$$F^k_t(e_j) = -f^k B^k_t(e_j).$$

$O^k_{s,t}(e_j)$ (for $t = 2, \dots, T+1$) principal outstanding at the beginning of time period t of debt type k issued in period s

$$O^k_{s,t}(e_j) = \begin{cases} B^k_{t-1}(e_j) & \text{for } s = t-1 \\ O^k_{s,t-1}(e_j) - R^k_{s,t-1}(e_j) & \text{for } s \neq t-1. \end{cases}$$

$IA^k_{s,t}(e_j)$ (for $t = 2, \dots, T+1$) interest accrued at beginning of period t for debt type k issued in period s:

for k a type of bond

$$IA^k_{s,t}(e_j) = IA^k_{s,t-1}(e_j) + IP^k_{s,t-1}(e_j) + r^k_{s,s}(e_j) O^k_{s,t}(e_j)$$

for k a type of short-term debt

$$IA^k_{s,t}(e_j) = IA^k_{s,t-1}(e_j) + IP^k_{s,t-1}(e_j) + r^k_{s,t-1}(e_j) O^k_{s,t}(e_j).$$

$IP^k_{s,t}(e_j)$ interest paid at beginning of period t for debt type k issued in period s

$$IP^k_{s,t}(e_j) = \begin{cases} -IA^k_t(e_j) & \text{if } k \text{ pays interest in period } t \\ 0 & \text{otherwise.} \end{cases}$$

$RC^k_{s,t}(e_j)$ retirement costs in period t for debt type k issued in period s and retired in period t

$$RC^k_{s,t}(e_j) = -g^k_{s,t}(e_j) R^k_{s,t}(e_j).$$

$EV^k_{s,T+1}(e_j)$ market value at the end of the planning period for debt type k issued in period s:

for k a callable bond

$$EV^k_{s,T+1}(e_j) = \frac{r^k_{s,s}(e_j)O^k_{s,T+1}(e_j)}{r^k_{s,T+1}(e_j)}\left[1 - \frac{1}{(1 + r_{s,T+1}(e_j)^k)}ry^k_{s,t}\right] + \frac{O^k_{s,T+1}(e_j)}{(1 + r_{s,T+1}(e_j)^k)}ry^k_{s,t} + IA^k_{s,T+1}(e_j)$$

for k a noncallable bond

$$EV^k_{s,T+1}(e_j) = \frac{r^k_{s,s}(e_j)O^k_{s,T+1}(e_j)}{r^k_{s,T+1}(e_j)}\left[1 - \frac{1}{(1 + r_{s,T+1}(e_j)^k)}r^k_{s,T+1}\right] + \frac{O^k_{s,T+1}(e_j)}{(1 + r_{s,T+1}(e_j)^k)}r^k_{s,t} + IA^k_{s,T+1}(e_j)$$

for k a type of short-term credit

$$EV^k_{s,T+1}(e_j) = O^k_{s,T+1} + IA^k_{s,T+1}(e_j).$$

$SFC^k_{s,t}(e_j)$ sinking fund contributions at the beginning of period t for debt type k issued in period s

$$SFC^k_{s,t}(e_j) = -sfc^k_{s,t}O^k_{s,t}(e_j).$$

$ASFE^k_{s,t}(e_j)$ accrued sinking fund earnings in period t for debt type k issued in period s

$$ASFE^k_{s,t}(e_j) = ASFE^k_{s,t-1}(e_j) + rsf^k_t(e_j)\, SF^k_{s,t}(e_j) - SFE^k_{s,t-1}(e_j).$$

$SFE^k_{s,t}(e_j)$ sinking fund earnings received in period t for debt type k issued in period s

$$SFE^k_{s,t}(e_j) = ASFE^k_{s,t}(e_j).$$

$SFW^k_{s,t}(e_j)$ sinking fund withdrawals in period t for debt type k issued in period s

$$SFW^k_{s,t}(e_j) = [SF^k_{s,t}(e_j) + SFC^k_{s,t}(e_j) + SFE^k_{s,t}(e_j)]\, R^k_{s,t}(e_j)/O^k_{s,t}(e_j).$$

$SF^k_{s,t}(e_j)$ (for $t = 2, \ldots, T+1$) sinking fund balance at the beginning of period t for debt type k issued in period s

$$\begin{aligned} SFW^k_{s,t}(e_j) =& SF^k_{s,t-1}(e_j) + SFC^k_{s,t-1}(e_j) \\ &+ SFE^k_{s,t-1}(e_j) - SFW^k_{s,t-1}(e_j). \end{aligned}$$

$NO^k_{s,t}(e_j)$ net principal outstanding at the beginning of period t for debt type k issued in period s:

for k a type of sinking fund bond

$$NO^k_{s,t}(e_j) = O^k_{s,t}(e_j) - SF^k_{s,t}(e_j)$$

for k not a type of sinking fund bond

$$NO^k_{s,t}(e_j) = O^k_{s,t}(e_j).$$

$EVSF^k_{s,T+1}(e_j)$ market value at the end of the planning period for the sinking fund for debt type k issued in period s

$$EVSF^k_{s,T+1}(e_j) = SF^k_{s,T+1}(e_j).$$

$NEV^k_{s,T+1}(e_j)$ net market value at the end of the planning period for debt type k issued in period s:

for k a type of sinking fund bond:

$$NEV^k_{s,T+1}(e_j) = EV^k_{s,T+1}(e_j) - EVSF^k_{s,T+1}(e_j)$$

for k not a type of sinking fund bond:

$$NEV^k_{s,T+1}(e_j) = EV^k_{s,T+1}(e_j).$$

$TCF^k_{s,t}(e_j)$ (for $t = 1, \ldots, T+1$) total cash flows in period t for debt type k issued in period s

$$TCF^k_{s,t}(e_j) = \begin{cases} B^k_s(e_j) + F^k_s(e_j) & \text{for } s = t \\ IP^k_{s,t}(e_j) + RC^k_{s,t}(e_j) + R^k_{s,t}(e_j) & \text{for } s \neq t \text{ and} \\ \quad +SFC^k_{s,t}(e_j) + SFW^k_{s,t}(e_j) & \quad t < T+1 \\ EV^k_{s,t}(e_j) & \text{for } t = T+1. \end{cases}$$

$TOCF^k_{s,t}(e_j)$ total operating cash outflows in period t for debt type k issued in period s

$$TOCF^k_{s,t}(e_j) = \begin{cases} IP^k_{s,t}(e_j) + SFC^k_{s,t}(e_j) & \text{for } s \neq t \\ 0 & \text{for } s = t. \end{cases}$$

In Canadian dollars:

$C\$OP^k_{s,t}(e_j)$ For each output $OP^k_{s,t}(e_j)$ above, the Canadian dollar equivalent:

$$C\$OP^k_{s,t} = \rho^k_t OP^k_{s,t}.$$

1.2.2 Performance indicators for a single event sequence (e_j)

$NPV^k_s(e_j)$ net present value of cash flows at time $t = 1$ for debt type k issued in period s, discounted at the borrower's discount rate

$$NPV^k_s(e_j) = \sum_{t=1}^{T+1} TCF^k_{s,t}(e_j)/(1+d)^{t-1}.$$

$IRR^k_s(e_j)$ internal rate of return for debt type k issued in period s

$$IRR^k_s(e_j) = \text{ discount rate } irr \text{ such that}$$
$$NPV^k_s(e_j) = \sum_{t=1}^{T+1} TCF^k_{s,t}(e_j)/(1+irr)^{t-1} = 0.$$

1.2.3 Expected values

$E(OP^k_{s,t})$ For each output $OP^k_t(e_j)$ above, the expected value over all event sequences

$$E(OP^k_{s,t}) = \sum_{j=1}^{J} p(e_j) OP^k_{s,t}(e_j).$$

2. Borrowing plan/portfolio model

For $j = 1,\ldots,J$, $s = 0,\ldots,t-1$, $t = 1,\ldots,T$, $m = 1,\ldots,M$, and $k = 1,\ldots,K$, unless otherwise noted, where k is a debt in the portfolio:

2.1 Input parameters

Various	for each member, all individual debt inputs and outputs listed above
C_t	borrower's cash requirement for period t. If negative, indicates an operating surplus.
M	maximum annual cash outflows for debt service during the planning period
N_t	maximum total borrowing over all debt types in period t
Q^k	maximum issue size of debt type k
q^k	minimum issue size of debt type k
CL^k	maximum principal outstanding for debt type k, where k is a type of short-term debt
U	maximum dollar amount of debt (at par) retired annually
L	minimum dollar amount of debt (at par) retired annually
S_1	cash surplus balance at the beginning of period 1
EAS_1	accrued earnings on the cash surplus balance at the beginning of period 1
i_t	mean or randomly-generated cash surplus earnings rate in period t
D_1	cash deficit (penalty borrowing) balance at the beginning of period 1
IAD_1	accrued interest on the cash deficit balance at the beginning of period 1
rd_t	mean or randomly-generated penalty borrowing rate in period t
NOTE:	All sums are of Canadian dollar equivalent amounts.

2.2 Output

2.2.1 Cash flows

$CSD_t(e_j)$ cash flow surplus or deficit for period t

$$CSD_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} TCF_{s,t}(e_j) - C_t(e_j),$$

where k denotes a debt in the portfolio.

$SD_t(e_j)$ cash surplus deposits for period t

$$SD_t(e_j) = \begin{cases} \max(CSD_t(e_j) - DR_t(e_j), 0) & \text{if } CSD_t(e_j) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$SW_t(e_j)$ cash surplus withdrawals for period t

$$SW_t(e_j) = \begin{cases} \min(S_t(e_j) + SE_t(e_j), -CSD_t(e_j)) & \text{if } CSD_t(e_j) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

$EAS_t(e_j)$ earnings accrued in period t on cash surplus balance

$$EAS_t(e_j) = EAS_{t-1}(e_j) + i_{t-1}(e_j)S_t(e_j) - SE_{t-1}(e_j).$$

$SE_t(e_j)$ earnings received in period t on cash surplus balance

$$SE_t(e_j) = EAS_t(e_j).$$

$S_t(e_j)$ (for $t = 2, \ldots, T+1$) cash surplus balance at the beginning of period t

$$S_t(e_j) = S_{t-1}(e_j) + SD_{t-1}(e_j) + SE_{t-1}(e_j) - SW_{t-1}(e_j).$$

$EVS_{T+1}(e_j)$ ending market value of cash surplus balance at time $T+1$

$$EVS_{T+1}(e_j) = S_{T+1}(e_j).$$

$TCFS_t(e_j)$ total cash flows for cash surplus in period t

$$TCFS_t(e_j) = SW_{t-1}(e_j) - SD_{t-1}(e_j).$$

$DB_t(e_j)$ cash deficit (penalty) borrowing in period t

$$DB_t(e_j) = \begin{cases} \min(-CSD_t(e_j) - SW_t(e_j), 0) & \text{if } CSD_t(e_j) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

$DR_t(e_j)$ cash deficit (penalty borrowing) repayment in period t

$$DR_t(e_j) = \begin{cases} \min(D_t(e_j) + DI_t(e_j), CSD_t(e_j)) & \text{if } CSD_t(e_j) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$ADI_t(e_j)$ accrued interest in period t on the penalty borrowing balance

$$ADI_t(e_j) = ADI_{t-1}(e_j) + rd_{t-1}D_t(e_j) - DI_{t-1}(e_j).$$

$DI_t(e_j)$ interest paid in period t on the penalty borrowing balance

$$DI_t(e_j) = ADI_t(e_j).$$

$D_t(e_j)$ (for $t = 2, \ldots, T+1$) cash deficit balance at the beginning of period t

$$D_t(e_j) = D_{t-1}(e_j) + DB_{t-1}(e_j) - DR_{t-1}(e_j).$$

$TCFD_t(e_j)$ total cash flows for cash deficit in period t

$$TCFD_t(e_j) = DB_t(e_j) - DR_t(e_j) - DI_t(e_j).$$

$EVD_{T+1}(e_j)$ ending market value of cash deficit balance at time $T+1$

$$EVD_{T+1}(e_j) = D_{T+1}(e_j).$$

$B_t(e_j)$ total amount borrowed at the beginning of period t

$$B_t(e_j) = \sum_{k=1}^{K} B_t^k(e_j) + DB_t(e_j).$$

$F_t(e_j)$ total issue costs in period t

$$F_t(e_j) = \sum_{k=1}^{K} F_t^k(e_j).$$

$O_t(e_j)$ (for $t = 2, \ldots, T+1$) total debt principal outstanding at the beginning of time period t

$$O_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} O_{s,t}^k(e_j) + D_t(e_j).$$

$NO_t(e_j)$ (for $t = 2, \ldots, T+1$) net principal outstanding at the beginning of time period t

$$NO_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} NO_{s,t}^k(e_j) + S_t(e_j).$$

$IA_t(e_j)$ total accrued interest at the beginning of period t

$$IA_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} IA_{s,t}^k(e_j) + ADI_t(e_j).$$

$IP_t(e_j)$ total interest paid at the beginning of period t

$$IP_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} IP_{s,t}^k(e_j) + DI_t(e_j).$$

$RC_t(e_j)$ total retirement costs in period t

$$RC_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} RC_{s,t}^k(e_j).$$

$R_t(e_j)$ total principal retired in period t

$$R_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} R_{s,t}^k(e_j) + DR_t(e_j).$$

$EV_{T+1}(e_j)$ total debt market value at the end of the planning period

$$EV_{T+1}(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} EV_{s,T+1}^k(e_j) + EVD_{T+1}(e_j).$$

$NEV_{T+1}(e_j)$ net portfolio market value at the end of the planning period

$$NEV_{T+1}(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} NEV_{s,T+1}^{k}(e_j) + EVD_{T+1}(e_j) - EVS_{T+1}(e_j).$$

$SFC_t(e_j)$ total sinking fund contributions at the beginning of period t

$$SFC_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} SFC_{s,t}^{k}(e_j).$$

$ASFE_t(e_j)$ total sinking fund earnings accrued at the beginning of period t

$$ASFE_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} ASFE_{s,t}^{k}(e_j).$$

$SFE_t(e_j)$ total sinking fund earnings received in period t

$$SFE_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} SFE_{s,t}^{k}(e_j).$$

$SFW_t(e_j)$ total sinking fund withdrawals in period t

$$SFW_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} SFW_{s,t}^{k}(e_j).$$

$SF_t(e_j)$ (for $t = 1, \ldots, T+1$) total sinking fund balance at the beginning of period t

$$SF_t(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{t-1} SF_{s,t}^{k}(e_j).$$

$EVSF_{T+1}(e_j)$ total sinking fund market value at the end of the planning period

$$EVSF_{T+1}(e_j) = \sum_{k=1}^{K}\sum_{s=1}^{T} EVSF_{s,T+1}^{k}(e_j).$$

$TCF_t(e_j)$ (for $t = 1, \ldots, T+1$) total cash flows in period t

$$TCF_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{t-1} TCF_{s,t}^{k}(e_j) + TCFS_t^k(e_j) + TCFD_t^k(e_j).$$

$TOCF_t(e_j)$ total operating cash flows in period t

$$TOCF_t(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{T} TOCF_{s,t}^{k}(e_j) + DI_t^k(e_j).$$

2.2.2 Performance indicators

$NPVS(e_j)$ net present value of cash surplus cash flows at time $t = 1$, discounted at the borrower's discount rate

$$NPVS(e_j) = \sum_{t=1}^{T+1} TCFS_t(e_j)/(1+d)^{t-1}.$$

$NPVD(e_j)$ net present value of cash deficit cash flows at time $t = 1$, discounted at the borrower's discount rate

$$NPVD(e_j) = \sum_{t=1}^{T+1} TCFD_t(e_j)/(1+d)^{t-1}.$$

$NPV(e_j)$ net present value of cash flows at time $t = 1$, discounted at the borrower's discount rate

$$NPV(e_j) = \sum_{k=1}^{K} \sum_{s=1}^{s} NPV_s^k(e_j) + NPVS^k(e_j) + NPVD(e_j).$$

$IRR(e_j)$ internal rate of return

$$IRR(e_j) = \text{ discount rate } irr \text{ such that}$$
$$NPV(e_j) = \sum_{t=1}^{T+1} \left(TCF_t(e_j) + TCFS_t(e_j) + TCFD_t(e_j)\right) / (1 + irr)^{t-1} = 0.$$

$WAT_t(e_j)$ weighted average term in period t

$$WAT_t(e_j) = \frac{\sum_{k=1}^{K} \sum_{s=1}^{t-1} NO_{s,t}^k(e_j) \tau_{s,t}^k(e_j) + D_t(e_j) - S_t(e_j)}{\sum_{k=1}^{K} \sum_{s=1}^{s} NO_{s,t}^k(e_j) + D_t(e_j) - S_t(e_j)}.$$

$WAC_t(e_j)$ weighted average debt service in period t

$$WAC_t(e_j) = \frac{\sum_{k=1}^{K} \sum_{s=1}^{t-1} NO_{s,t}^k(e_j) TOCF_{s,t}^k(e_j) + DI_t(e_j) \cdot D_t(e_j)}{\sum_{k=1}^{K} \sum_{s=1}^{t-1} NO_{s,t}^k(e_j) + D_t(e_j) - S_t(e_j)}.$$

$\%ST_t(e_j)$ percent short-term in period t

$$\%ST_t(e_j) = \left[\frac{\sum_{l=1}^{L} \sum_{s=1}^{t-1} NO_{s,t}^l(e_j) + D_t(e_j) - S_t(e_j)}{\sum_{k=1}^{K} \sum_{s=1}^{t-1} O_{s,t}^k(e_j) + D_t(e_j) - S_t(e_j)} \right] (100.0)$$

for l a debt with remaining term ≤ 1.

$\%MT_t(e_j)$ percent medium-term in period t

$$\%MT_t(e_j) = \left[\frac{\sum_{l=1}^{L} \sum_{s=1}^{t-1} NO_{s,t}^l(e_j)}{\sum_{k=1}^{K} \sum_{s=1}^{t-1} NO_{s,t}^k(e_j)} \right] (100.0)$$

for l a debt with remaining term > 1 and ≤ 15.

$\%LT_t(e_j)$ percent long-term in period t

$$\%LT_t(e_j) = \left[\frac{\sum_{l=1}^{L}\sum_{s=1}^{t-1} NO_{s,t}^{l}(e_j)}{\sum_{k=1}^{K}\sum_{s=1}^{t-1} NO_{s,t}^{k}(e_j)}\right](100.0)$$

for l a debt with remaining term > 15.

$\%VT_t(e_j)$ percent variable-rate in period t

$$\%VT_t(e_j) = \left[\frac{\sum_{l=1}^{L}\sum_{s=1}^{t-1} NO_{s,t}^{l}(e_j) + D_t(e_j) - S_t(e_j)}{\sum_{k=1}^{K}\sum_{s=1}^{t-1} NO_{s,t}^{k}(e_j) + D_t(e_j) - S_t(e_j)}\right](100.0)$$

for l a debt with variable interest rate.

2.2.3 Constraint satisfaction indicators

$MC_t(e_j)$ maximum cash outflows for debt service in period t

$$MC_t(e_j) = M - TOCF_t(e_j).$$

$MB_t(e_j)$ maximum total borrowing in period t

$$MB_t(e_j) = N_t - B_t(e_j).$$

$MX_t^k(e_j)$ maximum issue size for debt type k in period t

$$MX_t^k = Q^k - B_t^k(e_j).$$

$MN_t^k(e_j)$ minimum issue size for debt type k in period t

$$MN_t^k = q^k - B_t^k(e_j).$$

$MCL_{s,t}^k(e_j)$ short-term credit line, for k a type of short-term debt

$$MCL_{s,t}^k(e_j) = CL^k - O_{s,t}^k(e_j).$$

$MXR_t(e_j)$	maximum principal retired in period t

$$MXR_t(e_j) = U - R_t(e_j).$$

$MNR_t(e_j)$	minimum principal retired in period t

$$MNR_t(e_j) = L - R_t(e_j).$$

2.2.4 Expected values

$E(OP_t)$	For each portfolio output $OP_t(e_j)$ above, its expected value over all event sequences

$$E(OP_t) = \sum_{j=1}^{J} p(e_j)\, OP_t(e_j).$$

Appendix C

MIDAS Design Specification

Object-oriented systems differ from traditional systems in that they integrate rather than separate data and processes in their conceptual design and implementation. Structured tools which focus on either data or processes individually cannot, therefore, document object-oriented systems without modification.

The design specifications for MIDAS are presented here using a combination of modified dataflow diagrams, object class hierarchy charts and detailed object and method definitions. The documentation approach is based on commonly-accepted tools and conventions (Gane and Sarson 1979; Whitten, Bentley and Barlow 1989), modified for this project. Graphic documentation was produced using Excelerator CASE software (Index Technology Corporation 1989).

System documentation consists of the following components:

1. *System overview diagram.* This diagram decomposes the system into groups and subgroups of objects related by function. It gives an overview of system architecture and documentation organization; each connected level in the tree corresponds to an object diagram.

2. *Object diagrams.* Object diagrams show object groups in functional relationships, connected by both data flows and message (method call) flows. These diagrams modify Gane and Sarson dataflow diagrams to (a) combine processes and data stores into objects rather than showing them as separate system components, and (b) show message flows as well as data flows within the system.

3. *Class diagrams.* These show the inheritance class relationships used to define object classes in the system's knowledge base. They further document some

object groups in the lowest-level object diagrams.

4. *Object definitions.* These give detailed definitions of object classes and instances in the system knowledge base. They further document (a) object classes in the class hierarchy charts, and (b) objects in the lowest-level object diagrams which are not defined in class hierarchies.

5. *Method structure charts.* These decompose complex methods (named in object definitions) into modules and show their control structures.

6. *Primitive method specifications.* These specify processing logic for primitive methods named in object definitions.

7. *Rule specifications.* Individual rules are stated in an English-like syntax, and each set of rules is documented as to its purpose and chaining sequence.

Input and output formats are not given here but are illustrated in Chapter 10.

Specific conventions for each type of documentation are described in detail in the applicable sections of this Appendix.

For the portions of the design which are implemented, this documentation is supplemented by the system code listings provided on microfiche. These contain all LISP code implementing the prototype system and give additional detail on object definitions, specific method calls and data flow contents.

This documentation covers optimization, plan refinement and simulation, with related system support and user support, for single-path rate scenarios. Extension to branching scenarios would rquire additional methods, method arguments and summary objects to create, identify and manage multiple worlds.

1. System overview diagram

Figure C.1 gives the system overview diagram.

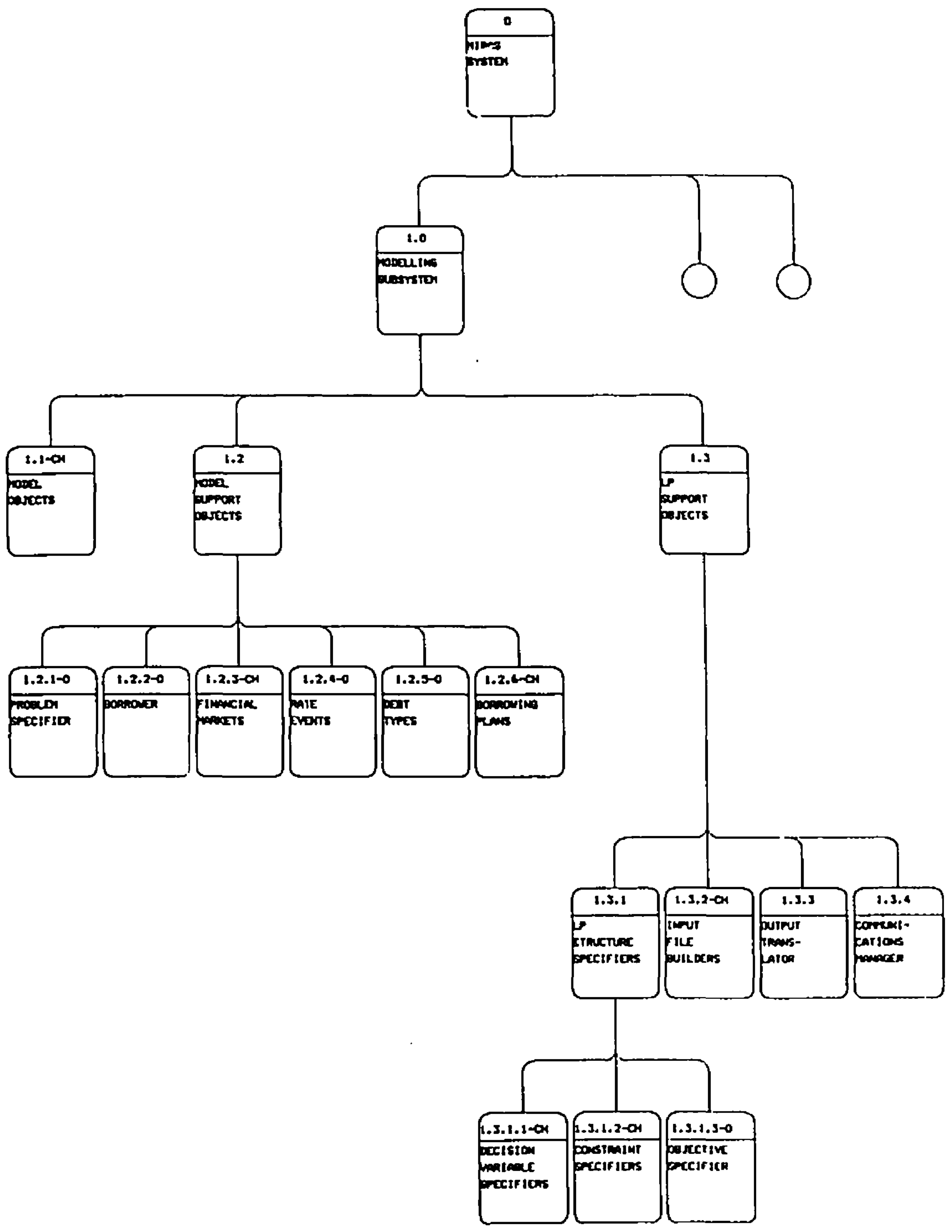


Figure C.1. MIDAS System Overview

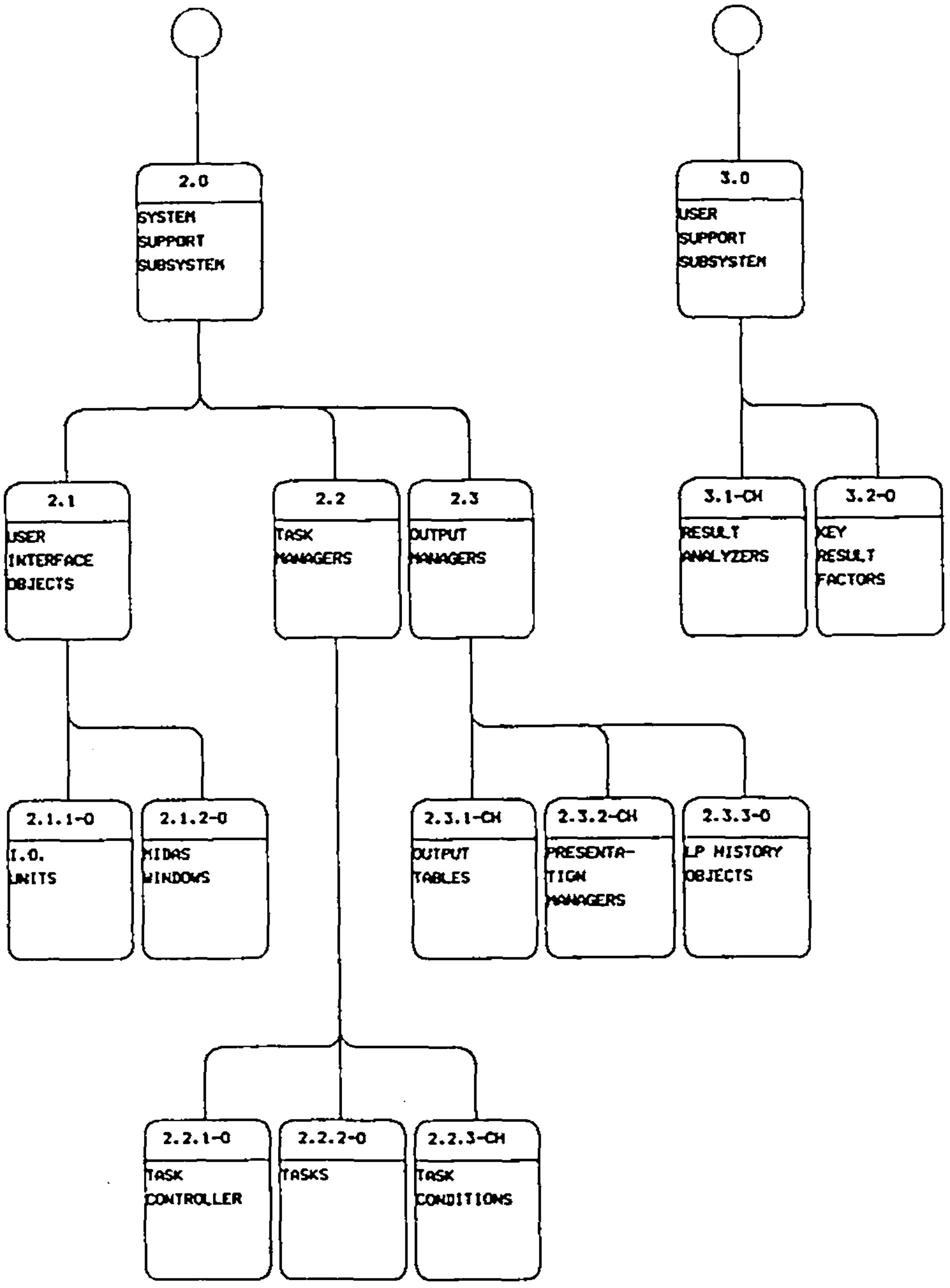
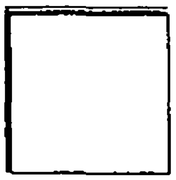


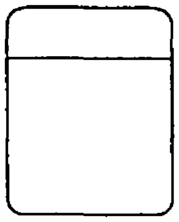
Figure C.1. MIDAS System Overview (Contd.)

2. Object diagrams

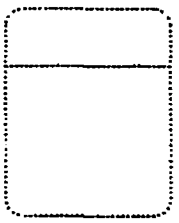
Obiect diagrams use the following symbols and conventions:

1.  A *square* represents an input source or output destination outside the system.

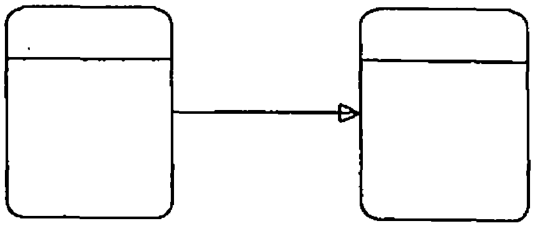
2. A *rounded rectangle* represents a subsystem, object group or primitive object, as follows:

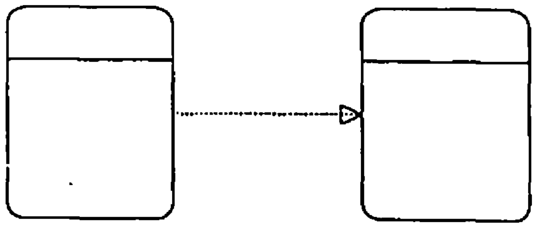


A *solid-line rounded rectangle* represents a subsystem, object subgroup or object within that group.



A *dashed-line rounded rectangle* represents a subsystem, object subgroup or object *outside* the object group being described in the current diagram.

3.  A *solid arrow* represents a data flow between objects or between objects and external entities.

4.  A *dashed arrow* represents a message flow (processing request and accompanying parameters) between objects.

5. Beginning with the top level system diagram, object diagrams *explode to* lower-level object diagrams, class hierarchy diagrams or object definitions which give additional design details not provided in higher-level diagrams. Object groups which explode to class hierarchy charts are indicated by 'CH' in their identifying codes; objects which explode to object definitions are indicated by 'O' in their identifying codes.

Because all object diagrams in this documentation were used in the text of this disssertation, they are referenced here rather than duplicated. Their locations are as follows:

1. System architecture: objected-oriented view: Figure 4.6.
2. Modelling subsystem: Figure 6.1.
3. Modelling subsystem: model object detail: Figure 6.2.
4. Modelling subsystem: model support object detail: Figure 6.6.
5. Modelling subsystem: LP support model object detail: Figure 6.13.
6. System support subsystem: Figure 7.1.
7. System support subsystem: user interface detail: Figure 7.2.
8. System support subsystem: output management detail: Figure 7.5.
9. System support subsystem: task management detail: Figure 7.11.
10. User support subsystem: Figure 8.1.

3. Class hierarchy diagrams

Class hierarchy diagrams show the inheritance hierarchies defining object classes, subclasses and instances within the main MIDAS knowledge base.

Within these diagrams, rounded rectangles represent object classes or instances, solid connecting lines represent subclass relationships and dashed connecting lines represent member (instance) relationships.

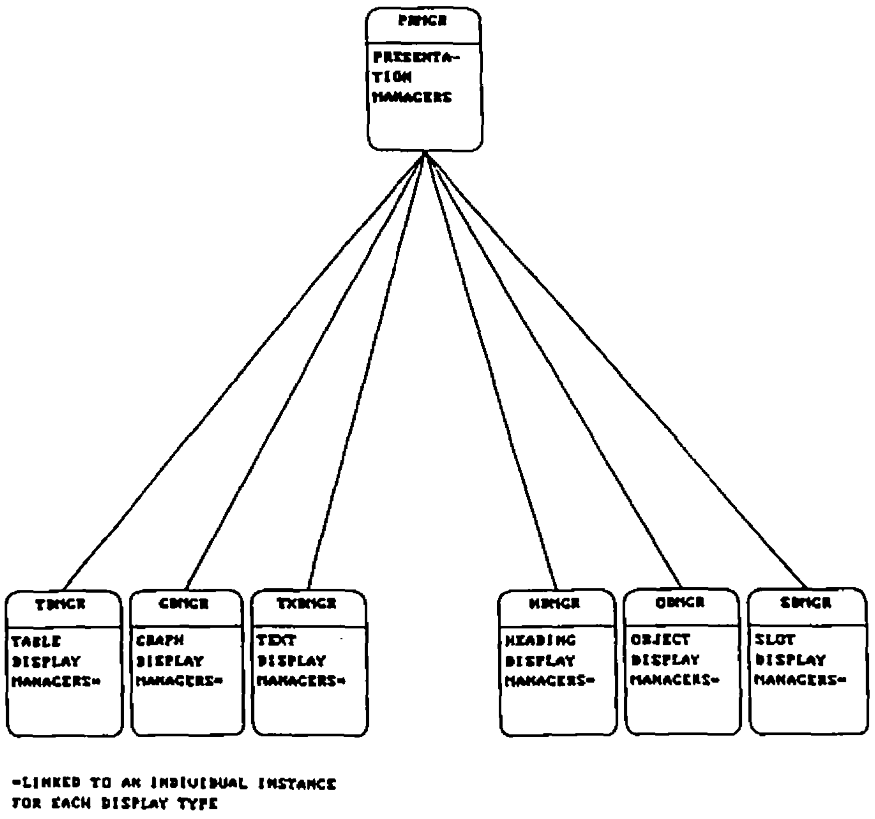


Figure C.2. Financial Markets Class Hierarchy

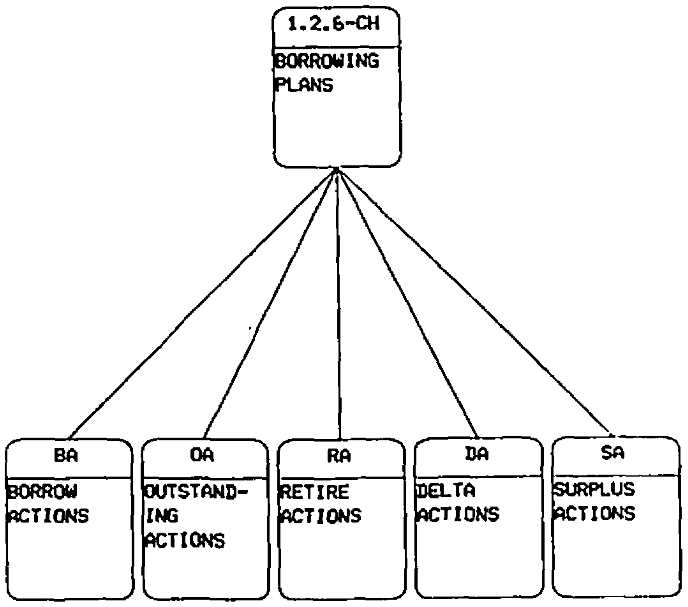


Figure C.3. Borrowing Plans Class Hierarchy

Class hierarchies used in MIDAS are the following:

1. Model.objects: Figure 6.3.
2. Financial.markets: Figure C.2.
3. Borrowing.plans: Figure C.3.
4. Decision.variable.specifiers: Figure 6.14.
5. Constraint.specifiers: Figure 6.17.
6. Input.file.builders: Figure C.4.
7. Task.conditions: Figure 7.14.
8. Presentation.managers: Figure C.5.
8. Result.analyzers: Figure 8.2.

4. Object definitions

An object definition is given for each class or individual (instance) object defined in the system knowledge base. An individual problem and its analysis results are represented in a separate knowledge base called a *problem space* which is stored separately from the main knowledge base. Objects in the problem space are created as instances of objects in the main knowledge base, so that the main knowledge base objects serve as templates for objects in the problem space. The object definitions given in this documentation are for classes and instance objects in the main knowledge base.

An object definition consists of the following:

1. NAME: The reference name within the knowledge base for the class or instance being defined.

2. SUBCLASS.OF: The name(s) of the parent class(es) of the object if the object is a class. Member slot and method names and values (see below) for subclasses are inherited from parent classes unless overridden; class objects pass their slots and methods to their child classes or instances. NIL indicates no parents for the class.

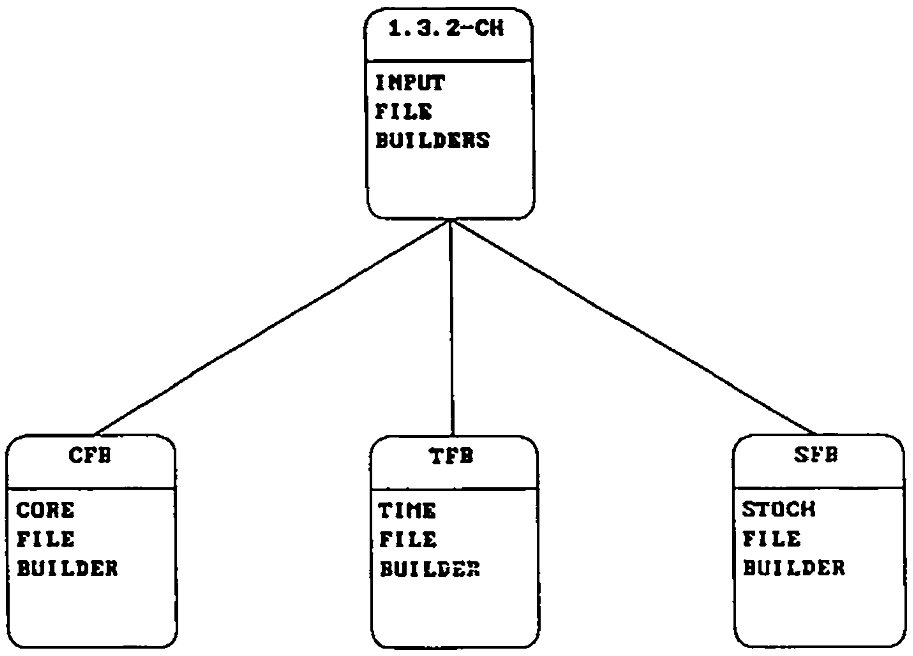


Figure C.4. Input File Builders Class Hierarchy

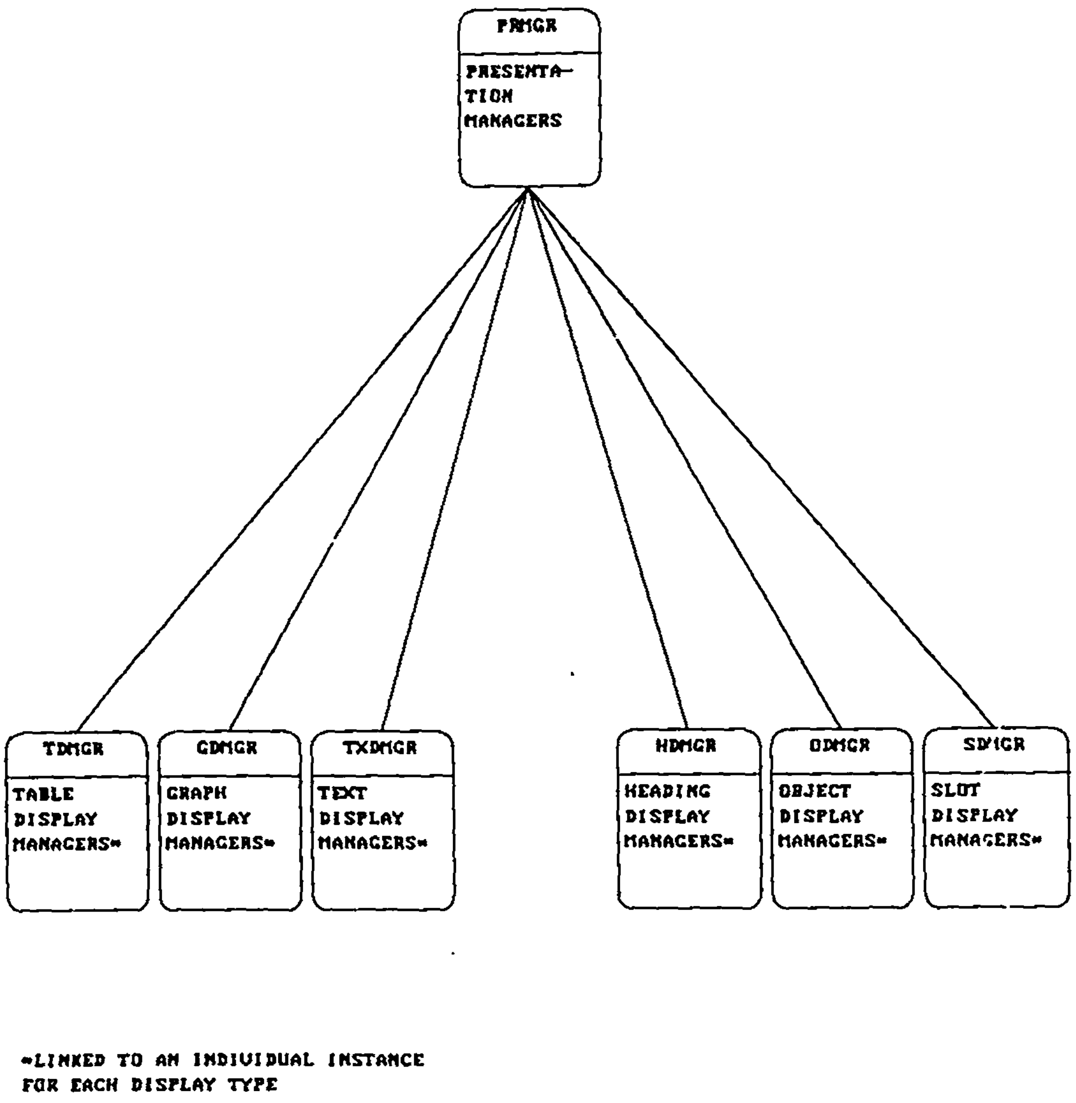


Figure C.5. Presentation Managers Class Hierarchy

3. INSTANCE.OF:	The name(s) of the parent class(es) for the object if the object is an instance. Slot and method names and values for instances are inherited from parent classes unless overridden. Instance classes do not have child classes or instances. NIL indicates no parents for the object.
4. SUBCLASSES:	The name(s) of subclasses of the object class in the main knowledge base. NIL indicates no such subclasses.
5. INSTANCES:	The name(s) of instances of the class defined in the main knowledge base. NIL indicates no such subclasses.
6. DESCRIPTION:	A comment field describing the purpose and role of the object.
7. ATTRIBUTE SLOTS:	The names and defining characteristics of slots which describe attributes of the object and which are inherited by its child objects. These slots are known as *member slots* in KEE. This section is omitted if there are no such slots with local definitions or values.
8. CLASS-SPECIFIC SLOTS:	The names and defining characteristics of slots which are not inherited by child objects. Own slots in class objects are attributes or methods pertaining to the class itself rather than to its members; all slots in instance objects are own slots. These slots are known as *own slots* in KEE. This section is omitted if there are no such slots with local definitions or values.
9. SLOT CHARACTERISTICS:	A slot may have *facets* which document, specify or limit the slot's value. Critical facets are listed for slots in object def-

initions; complete listings of slot facets are found in the knowledge base listing on microfiche. The facet ***inheritance*** specifies the way in which inheritance takes place for the slot; UNION means that values are combined in a set union operations so that values from parent classes are retained and added to by child classes, while unspecified inheritance means that values are overridden so that parent-class values are lost when new values are specified.

10. EXTERNAL-USE METHODS: The names of methods in the object which are called by methods in ***other objects***. (The prefix MTH. is used for method names in the system but has been omitted from all method slot names for clarity.) The ***value*** of a method slot is the name of the LISP function which defines the method's procedures. Comments describe methods in the objects in which they are first defined. If a value is given for a method, it is included in the system prototype and documented on microfiche; if it is marked with a $\star$, it is documented in §6 of this Appendix.

11. INTERNAL-USE METHODS: The names of methods in the object which are called only by methods ***within*** the object. (The prefix MTH. has been omitted from all method slot names for clarity.) The ***value*** of a method slot is the name of the LISP function which defines the method's procedures. Comments describe methods in the objects in which they are first defined. If a value is given for a method, it is included in the system prototype and documented in Appendix C; if it is marked with a $\star$, it is documented in §6 of this Appendix.

Object definitions are listed in depth-first order of the inheritance hierarchies in which they are defined, so that top-level object classes are presented first and followed by lower-level specializing classes.

4.1 Modelling subsystem

4.1.1 Model objects

```
NAME:          MODEL.OBJECTS
SUBCLASS.OF:   NIL
INSTANCE.OF:   NIL
SUBCLASSES:    FINANCIAL.INSTRUMENTS, PORTFOLIOS
INSTANCES:     NIL
DESCRIPTION:   Top-level class specifying attributes and method names
               common to both individual financial instruments (basic
               model objects) and portfolios (composite model objects).
ATTRIBUTE SLOTS:
      active.cf.slots
            comment  Names of slots in the associated CF.TABLE object which are
                     used by this model object during cash flow calculations.
      cf.table
            comment  Name of the CF.TABLE holding output for this object.
      class
            comment  The model object class for this object (inherited
                     by instances and used in displays and reports).
      unit.display.data
            comment  A slot to hold formatted data for display of details for
                     this object.
      unit.display.slots
            comment  A slot to hold the names of slots to be displayed in the
                     object detail display.
      unit.display.titles
            comment  A slot to hold titles for slot displays in the object detail display.
EXTERNAL-USE METHODS:
      add
            value    UNIT.MTH.ADD
            comment  Method to add an instance of this class of object in the
                     problem space.
      change
            value    UNIT.MTH.CHANGE
            comment  Method to change the user-specified slot values of an instance
                     of this class in the problem space.
      delete
            value    UNIT.MTH.DELETE
```

```
          comment   Method to delete an instance of this class from the
                    problem space.
     project
          comment   Top-level method controlling the cash flow projection process.
     simulate
          comment   Top-level method controlling random-rate simulation of
                    cash flows for instance of this class.
INTERNAL-USE METHODS:
     do.cfs
          comment   Method controlling cash flow initialization and calculation
                    sequence for an instance of this class.
     input
          comment   Method for obtaining form-based user input to define
                    an instance of this class.
```

```
NAME:          FINANCIAL.INSTRUMENTS
SUBCLASS.OF:   MODEL.OBJECTS
INSTANCE.OF:   NIL
SUBCLASSES:    INVESTMENTS, DEBTS
INSTANCES:     NIL
DESCRIPTION:   Highest basic model object class in hierarchy.
ATTRIBUTE SLOTS:
     currency
          comment   Currency of issue.
          default   C$
          range     Currencies in markets defined in the problem space.
     market
          comment   Market in which issued.
          default   CANADA
          range     Markets defined in the problem space.
     spread.slots
          comment   List of names of financial.market slots containing spreads
                    to be used in determining the market interest rate for
                    instances of this class.
          inheritance.  UNION
INTERNAL-USE METHODS:
     one.period.cfs
          comment   Method to calculate cash flows for instance of this class
                    for a single time period.
EXTERNAL-USE METHODS:
     do.cfs
          value     FI.MTH.DO.CFS
     project
          value     FI.MTH.PROJECT
     simulate
          value     FI.MTH.SIMULATE
```

```
NAME:          DEBTS
SUBCLASS.OF:   FINANCIAL.INSTRUMENTS
INSTANCE.OF:   NIL
SUBCLASSES:    LONG.TERM.DEBTS, SHORT.TERM.DEBTS
INSTANCES:     NIL
DESCRIPTION:   Generic debt definition.
ATTRIBUTE SLOTS:
    accrued.interest
        comment   Accrued interest balance for an instance of this class as at
                  the knowledge base date, which should correspond to the start
                  of the planning period.
        default   0.0
    active.cf.slots
        values    CF.PRINCIPAL.BORROWED, CF.PRINCIPAL.RETIRED,
                  CF.INTEREST.ACCRUED, CF.INTEREST.PAID,
                  CF.INTEREST.RATE, CF.REMAINING.TERM,
                  CF.STATUS, CF.TOTAL.CASH.FLOWS,
                  CF.TOTAL.OPERATING.CASH.FLOWS,
                  CF.NET.CASH.FLOWS, CF.NET.OPERATING.CASH.FLOWS
    principal.outstanding
        comment   Outstanding balance for instance of this instrument
                  as at the knowledge base date.
        default   0.0
    term
        comment   Length of time in years instrument is outstanding.
        default   0.0
```

```
NAME:          LONG.TERM.DEBTS
SUBCLASS.OF:   DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    BONDS
INSTANCES:     NIL
DESCRIPTION:   Generic long-term debt definition.
ATTRIBUTE SLOTS:
    original.principal
        comment   Amount for which an instance of this class was or is to be
                  originally issued. Used in cash flow projections when the
                  object is to be issued during the projection period.
        default   0.0
```

```
NAME:          BONDS
SUBCLASS.OF:   LONG.TERM.DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    BASIC.BONDS, CALLABLE.BONDS, FOREIGN.BONDS,
               SINKING.FUND.BONDS
INSTANCES:     NIL
```

DESCRIPTION: Bond class definition (specialization of long-term debts).

ATTRIBUTE SLOTS:

active.cf.slots		
	values	CF.ISSUE.COSTS, CF.RETIREMENT.COST
class		
	value	BOND
coupon.rate		
	comment	Contract interest rate at which instance of this class is issued.
	default	0.0
interest.payments.per.year		
	comment	Number of interest payments required by bond contract.
	default	2
	range	1, 2 or 4
issue.admin.costs		
	comment	Dollar amount of administrative costs on issue. Part of total issue costs.
	default	0.0
issue.commission.rate		
	comment	Per-dollar rate for commission costs on issue costs. Part of total issue costs.
	default	0.0
issue.date		
	comment	Date of issue, in Explorer universal date format (time since Dec 31, 1899).
	default	0
issue.discount.or.premium		
	comment	Dollar amount of discount or premium on issue. Assumed zero for new issues. Computed from public price.
issue.period		
	comment	Index of time period of issue, relative to beginning of planning period. Computed at start of cf projection or simulation.
issue.quarter		
	comment	Quarter of issue. Computed from issue.date.
	range	1, 2, 3 or 4
issue.total.costs		
	comment	Dollar amount of total issue costs. Either input directly or computed from admin costs and discount or premium.
	default	0.0
issue.year		
	comment	Year of issue. Computed from issue date.
maturity.date		
	comment	Date bond issue matures (must be retired), according to contract. Computed from issue date and term.
maturity.period		
	comment	Index of time period of maturity, relative to beginning of planning period. Computed at start of cf projection or simulation.
maturity.quarter		
	comment	Quarter in which issue matures. Computed from

```
                    maturity date.
    maturity.year
         comment    Year in which issue matures. Computed from maturity date.
    quarters.interest.paid
         comment    List of quarters in which interest payments required.
    spread.slots
         value      SPREAD.BOND
INTERNAL-USE METHODS:
    cf.carry.fwd
         comment    Method to compute beginning balance for a cash
                    flow period.
         value      BONDS.MTH.CF.CARRY.FWD
    cf.end.value
         comment    Method to compute market value for an instance of
                    this class at the end of the planning period.
         value      BONDS.MTH.CF.END.VALUE
    cf.interest
         comment    Method to compute interest accrued and paid for
                    a single time period.
         value      BONDS.MTH.CF.INTEREST
    cf.issue
         comment    Method to compute cash flows for a bond issue.
         value      BONDS.MTH.CF.ISSUE
    cf.retire
         comment    Method to compute cash flows on retirement.
         value      BONDS.MTH.CF.RETIRE
    cf.setup
         comment    Method to set up opening balance at the beginning
                    of the planning period.
         value      BONDS.MTH.CF.SETUP
    cf.status
         comment    Method to determine and record in the bond's cf.table the status
                    indicators for the bond (NEW, ACTIVE, MATURE, CALLABLE,
                    CALLED) for each time period during the planning period.
         value      BONDS.MTH.CF.STATUS
    dates
         comment    Method to calculate issue and maturity years and quarters
                    from input dates.
         value      BONDS.MTH.CF.DATES
    one.period.cfs
         value      BONDS.MTH.ONE.PERIOD.CFS
    update.features
         comment    Control method for input of parameters for bond features
                    (call options, foreign currency, and sinking funds).
         value      BONDS.MTH.UPDATE.FEATURES
```

```
NAME:          BASIC.BONDS
SUBCLASS.OF:   BONDS
INSTANCE.OF:   NIL
```

```
SUBCLASSES:   NIL
INSTANCES:    NIL
DESCRIPTION:  Lowest-level bond class with no special features.
              All slots and methods inherited from bonds. Included for symmetry with specialize
```

```
NAME:         CALLABLE.BONDS
SUBCLASS.OF:  BONDS
INSTANCE.OF:  NIL
SUBCLASSES:   NIL
INSTANCES:    NIL
DESCRIPTION:  Definitional class for bonds with a call feature.
ATTRIBUTE SLOTS:
      call.first.call.price
            comment   Call price in the first year in which the issue is callable.
            comment   First year in which the issue is callable.
      call.part?
            comment   Indicator as to whether the issue can be called in part.
            default   NIL (indicating NO)
      call.prices
            comment   List of call prices in format ((year price)...).
                      Computed from first call price and first call year.
      spread.slots
            value     SPREAD.CALL
INTERNAL-USE METHODS:
      call.option.input
            comment   Input method for call option parameters.
            value     CALLABLE.BONDS.MTH.CALL.OPTION.INPUT
      call.prices
            comment   Method to compute call prices and set up list.
            value     CALLABLE.BONDS.MTH.CALL.PRICES
      cf.call
            comment   Method to compute cash flows in a period in which the issue
                      is called.
            value     CALLABLE.BONDS.MTH.CF.CALL
      call.price
            comment   Method to return the call price for an issue for a specified year.
```

```
NAME:         FOREIGN.BONDS
SUBCLASS.OF:  BONDS
INSTANCE.OF:  NIL
SUBCLASSES:   NIL
INSTANCES:    NIL
DESCRIPTION:  Bonds in foreign currency.
INTERNAL-USE METHODS:
      fgn.input
            comment   Method to obtain input for foreign bond slots.
```

```
        value      FGN.BONDS.MTH.FGN.INPUT
```

```
NAME:          SINKING.FUND.BONDS
SUBCLASS.OF:   BONDS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for bonds with sinking fund feature.
ATTRIBUTE SLOTS:
    active.cf.slots
        values     SF.BALANCE, SF.CONTRIBUTIONS,
                   SF.EARNINGS.ACCRUED, SF.EARNINGS,
                   SF.EARNINGS.RATE, SF.STATUS,
                   SF.TOTAL.CASH.FLOWS, SF.TOTAL.EARNINGS,
                   SF.WITHDRAWALS
    sf.name
        comment    The name of the sinking fund object for an instance
                   of this class.
    spread.slots
        value      SPREAD.SF
INTERNAL-USE METHODS:
    sf.input
        value      SINKING.FUND.BONDS.MTH.INPUT
```

```
NAME:          SHORT.TERM.DEBTS
SUBCLASS.OF:   DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    BANK.CREDIT, PENALTY.CREDIT, PROVINCIAL.CREDIT
INSTANCES:     NIL
DESCRIPTION:   Generic short-term debt class. Instances represent
               sources of short-term credit.
ATTRIBUTE SLOTS:
    class
        value      SHORT.TERM.DEBT
    credit.maximum
        comment    Maximum credit available from this source.
INTERNAL-USE METHODS:
    cf.borrow
        comment    Method to compute cash flows for borrowing from this
                   source, based on borrow actions for this source in the current
                   borrowing plan.
        value      STD.MTH.CF.BORROW
    cf.carry.fwd
        comment    Method to compute beginning balance for a cash flow period.
        value      STD.MTH.CF.CARRY.FWD
    cf.end.value
```

```
          comment  Method to compute market value for an instance of this
                   class at the end of the planning period.
          value    STD.MTH.CF.END.VALUE
     cf.interest
          comment  Method to compute interest accrued and paid for a
                   single time period.
          value    STD.MTH.CF.INTEREST
     cf.retire
          comment  Method to compute cash flows on retirement, based on
                   retire actions for this source in the current borrowing plan.
          value    STD.MTH.CF.RETIRE
     cf.setup
          comment  Method to set up opening balance at the beginning of the
                   planning period.
          value    STD.MTH.CF.SETUP
     cf.status
          comment  Method to determine and record in the bond's cf.table
                   the status indicators for the debt sourcs (ACTIVE, NIL)
                   for each time period during the planning period.
          value    STD.MTH.CF.STATUS
     one.period.cfs
          value    STD.MTH.ONE.PERIOD.CFS
     input
          value    STD.MTH.INPUT
```

```
NAME:          BANK.CREDIT
SUBCLASS.OF:   SHORT.TERM.DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for bank credit (specialization of short-term debt).
ATTRIBUTE SLOTS:
     spread.slots
          value    SPREAD.BANK.CREDIT
```

```
NAME:          PENALTY.CREDIT
SUBCLASS.OF:   SHORT.TERM.DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for penalty credit (specialization of short-term debt).
ATTRIBUTE SLOTS:
     spread.slots
          value    SPREAD.PENALTY.CREDIT
```

```
NAME:          PROVINCIAL.CREDIT
SUBCLASS.OF:   SHORT.TERM.DEBTS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for provincial credit (specialization of short-term debt).
ATTRIBUTE SLOTS:
      spread.slots
           value       SPREAD.PROV.CREDIT
```

```
NAME:          INVESTMENTS
SUBCLASS.OF:   FINANCIAL.INSTRUMENTS
INSTANCE.OF:   NIL
SUBCLASSES:    LONG.TERM.INVESTMENTS, SHORT.TERM.INVESTMENTS
INSTANCES:     NIL
DESCRIPTION:   Generic investment class.
ATTRIBUTE SLOTS:
      accrued.earnings
           comment   Accrued earnings balance as at the knowledge base date,
                     which should correspond to the start of the planning period.
           default   0.0
      investment.balance
           comment   Balance for an instance of this class as at the knowledge
                     base date.
           default   0.0
```

```
NAME:          LONG.TERM.INVESTMENTS
SUBCLASS.OF:   INVESTMENTS
INSTANCE.OF:   NIL
SUBCLASSES:    SINKING.FUNDS
INSTANCES:     NIL
DESCRIPTION:   Generic long-term investment class. Included for completeness;
               all slots and methods inherited from parent classes.
```

```
NAME:          SINKING.FUNDS
SUBCLASS.OF:   LONG.TERM.INVESTMENTS
INSTANCE.OF:   NIL
SUBCLASSES:    C$.SINKING.FUNDS, FOREIGN.SINKING.FUNDS
INSTANCES:     NIL
DESCRIPTION:   Defining class for sinking funds (long-term investment funds
               driven by bond requirements).
ATTRIBUTE SLOTS:
      class
           value       SINKING.FUND
```

```
    debt.name
         comment   Name of bond object associated with an instance of
                   this class.
    sf.ceiling
         comment   Maximum required balance for an instance of this class
                   (often equal to the bond face value).
    sf.contribution.amount
         comment   Dollar amount of required annual sinking fund contribution.
                   May be computed from contribution percent and bond
                   original principal.
         default   0.0
    sf.contribution.percent
         comment   Percentage of bond original principal to be contributed
                   annually to the sinking fund.
         default   0.0
    sf.start.year
         comment   Year in which sinking fund contributions are to begin.
    spread.slots
         value     SPREAD.SF.EARNINGS
EXTERNAL-USE METHODS:
    one.period.cfs
         value     SF.MTH.ONE.PERIOD.CFS
INTERNAL-USE METHODS:
    cf.carry.fwd
         comment   Method to compute the beginning sinking fund balance
                   for a cash flow period.
         value     SF.MTH.CF.CARRY.FWD
    cf.contributions
         comment   Method to compute the cash flows for a contribution
                   to the sinking fund for a single time period.
         value     SF.MTH.CF.CONTRIBUTIONS
    cf.earnings
         comment   Method to compute the earnings accrued and received
                   for a sinking fund for a single time period.
         value     SF.MTH.CF.EARNINGS
    cf.end.value
         comment   Method to compute the value for a sinking fund
                   at the end of the planning period.
         value     SF.MTH.CF.END.VALUE
    cf.setup
         comment   Method to set up the balance of an existing sinking
                   fund at the beginning of a planning period.
         value     SF.MTH.CF.SETUP
    cf.status
         comment   Method to determine the status of a sinking fund
                   (ACTIVE, CONTRIBUTE, WITHDRAW, NIL)
                   for a single time period.
         value     SF.MTH.CF.STATUS
    cf.withdrawals
         comment   Method to compute the cash flows associated with
```

```
                    a sinking fund withdrawal on bond retirement.
          value     SF.MTH.CF.WITHDRAWALS
     input
          value     SF.MTH.INPUT
```

```
NAME:          C$.SINKING.FUNDS
SUBCLASS.OF:   SINKING.FUNDS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Class for sinking funds in domestic currency. All attributes,
               methods and values inherited from generic sinking fund class.
```

```
NAME:          FOREIGN.SINKING.FUNDS
SUBCLASS.OF:   SINKING.FUNDS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for sinking fund in foreign currency. All slots
               and methods are inherited from parent class, but cash flow calculations
               check for parent class and translate fx if foreign.
```

```
NAME:          SHORT.TERM.INVESTMENTS
SUBCLASS.OF:   INVESTMENTS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Generic short-term investment class. Now used only for the
               portfolio CASH.SURPLUS.
ATTRIBUTE SLOTS:
     active.cf.slots
          value     CF.REMAINING.TERM
     class
          value     SHORT.TERM.INVESTMENT
     spread.slots
          value     SPREAD.BANK.DEPOSIT
INTERNAL-USE METHODS:
     cf.carry.fwd
          comment   Method to compute beginning balance for instance
                    of this class for a single time period.
          value     STC.MTH.CF.CARRY.FWD
     cf.end.value
          comment   Method to compute the value of an instance of this
                    class at the end of a planning period.
```

```
        value     STC.MTH.CF.END.VALUE
    cf.interest
        comment   Method to compute the cash flows for earnings accrued and
                  received for a single time period for an instance of this class.
        value     STC.MTH.INTEREST
    cf.invest.or.withdraw
        comment   Method to computer the cash flows for investing
                  or withdrawing from an instance of this class, based on
                  surplus actions in the current borrowing plan.
        value     STC.MTH.INVEST.OR.WITHDRAW
    cf.setup
        comment   Method to set up opening balance for an instance
                  of this class at the start of a planning period.
        value     STC.MTH.SETUP
    cf.status
        comment   Method to determine the status (ACTIVE or NIL)
                  for an instance of this class for a single time period.
        value     STC.MTH.CF.STATUS
    input
        value     STC.MTH.INPUT
    one.period.cfs
        value     STC.MTH.ONE.PERIOD.CFS
```

```
NAME:          PORTFOLIOS
SUBCLASS.OF:   MODEL.OBJECTS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Generic portfolio definition, containing slots and methods for linking
               individual financial instruments and handling cash surpluses and deficits.
ATTRIBUTE SLOTS:
    active.cf.slots
        value     CF.CASH.NEEDS, CF.CASH.DEFICIT,
                  CF.AVERAGE.DEBT.SVC, CF.AVERAGE.TERM,
                  CF.PERCENT.LT, CF.PERCENT.MT, CF.PERCENT.ST,
                  CF.PERCENT.VARIABLE
    class
        value     PORTFOLIO
    pf.markets
        comment   List of markets in which this portfolio has debts.
        range     Markets defined in this problem space.
    pf.members
        comment   List of financial instruments which make up this portfolio.
        range     Financial instruments defined for this problem space.
EXTERNAL-USE METHODS:
    optimize*
        comment   Top-level control method for plan and portfolio optimization.
    project
```

```
        value      PF.MTH.PROJECT
    simulate
        value      PF.MTH.SIMULATE
INTERNAL-USE METHODS:
    borrow.st
        comment    Method to create a borrow action to cover the portfolio's
                   cash deficit for a single time period.
        value      PF.MTH.BORROW.ST
    cash.deficit
        comment    Method to handle cash deficits and surpluses with respect to
                   cash requirements during cash flow projection and simulation.
        value      PF.MTH.CASH.DEFICIT
    do.cfs
        value      PF.MTH.DO.CFS
    configure.optimization.model*
        comment    Method to control sequence of operations for LP formulation.
    input
        value      PF.MTH.INPUT
    invest.st
        comment    Method to create a surplus action to invest a cash surplus
                   for a single time period.
        value      PF.MTH.INVEST.ST
    retire.st
        comment    Method to create a retire short-term debt
                   when there is a cash surplus in a single time period.
        value      PF.MTH.RETIRE.ST
    withdraw.st
        comment    Method to create a surplus action to withdraw short-term
                   deposits to cover a cash deficit in a single time period.
        value      PF.MTH.WITHDRAW.ST
```

4.1.2 Model support objects

```
NAME:           PROBLEM.SPECIFIERS
SUBCLASS.OF:    NIL
INSTANCE.OF:    NIL
SUBCLASSES:     NIL
INSTANCES:      NIL
DESCRIPTION:    Objects to hold and maintain high-level model parameters driving
                number of passes, time periods, planning horizon and names of objects
                currently in use.
ATTRIBUTE SLOTS:
    cf.begin.quarter
        comment    Quarter in which planning period begins.
        default    1
```

cf.begin.year		
	comment	Year in which planning period begins.
	default	1990
cf.col.headings		
	comment	List of formatted year and quarter headings for use in table and graph displays. Generated by table and graph display managers.
cf.col.titles		
	comment	List containing 'year' and 'quarter' headings for use in displays. Generated by table and display managers.
cf.end.quarter		
	comment	Quarter in which planning period ends.
	default	4
cf.end.year		
	comment	Year in which planning period ends.
	default	2000
cf.interval		
	comment	Length of time intervals making up the planning period.
	range	YEAR or QUARTER
	default	YEAR
cf.length		
	comment	Length of cash flow lists in output tables. Computed from the beginning and ending dates and length of time intervals.
cf.quarters		
	comment	List of quarter numbers for the planning period. Used in cash flow calculations.
cf.years		
	comment	List of years for the planning period. Used in cash flow calculations.
current.borrower		
	comment	Name of the borrower object currently in use.
	range	Borrowers defined for this problem space.
current.fi		
	comment	Name of the last financial instrument for which a cash flow projection or simulation was done.
	range	Names of financial instruments defined for the problem space.
current.pf		
	comment	Name of the last portfolio for which an optimization, simulation or cash flow projection was done.
	range	Names of portfolios defined in the problem space.
current.plan		
	comment	Name of the last borrowing plan for which an which a cash simulation or cash flow projection was done.
	range	Names of borrowing plans defined in the problem space.
knowledge.base.date		
	comment	Date of last update of opening balances.
rate.event.tree		
	comment	Name of the rate event tree to be used to generate future rate sequences.
	range	Names of rate event trees currently defined in the problem space.
sim.passes		

```
        comment   Number of passes to be used in stochastic in the
                  of portfolios or financial instruments.
EXTERNAL-USE METHODS:
    add
        value     PROBLEM.SPECIFIERS.MTH.ADD
    change
        value     PROBLEM.SPECIFIERS.MTH.CHANGE
```

Note: Problem specifiers have no delete method because one and only one problem specifier is required for each problem space

```
INTERNAL-USE METHODS:
    cf.dates
        comment   Method to build year and quarter lists from planning period
                  beginning and ending dates.
        value     PROBLEM.SPECIFIERS.MTH.CF.DATES
    cf.length
        comment   Method to compute and store length of cash flow lists
                  from planning period beginning and ending dates and
                  time interval specification.
        value     PROBLEM.SPECIFIERS.MTH.CF.LENGTH
    input
        value     PROBLEM.SPECIFIERS.MTH.INPUT
```

```
NAME:          BORROWERS
SUBCLASS.OF:   NIL
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for borrower objects, which handle goals, parameters and
               constraints set up by the borrower for whom plans are being modelled.
ATTRIBUTE SLOTS:
    cash.needs.actual.percent
        comment   Percentage to be used in converting budgeted
                  cash requirements to actual cash requirements.
        default   100.0
    cash.needs.budget
        comment   Budgeted cash requirements for an instance of this
                  class, in a list of the form ((year quarter amount)...)
    discount.rate
        comment   Discount rate to be used in calculating net present values.
    maximum.annual.debt.service
        comment   Dollar amount of maximum desired annual cash outflows
                  for debt service for an instance of this class.
    maximum.annual.borrowing
        comment   Dollar amount of maximum desired ammount borrowed
                  in a year.
    maximum.annual.retirement
        comment   Dollar amount of maximum desired debt amount retired
                  in a year.
```

minimum.annual.retirement

comment	Dollar amount of minimum desired debt amount retired in a year.

EXTERNAL-USE METHODS:

add

value	UNIT.MTH.ADD

cf.cash.needs

comment	Method to generate lists of actual cash requirements in pf.tables from the budgeted cash requirements defined for an instance of this class.
value	BORROWERS.MTH.CF.CASH.NEEDS

change

value	UNIT.MTH.CHANGE

delete

value	UNIT.MTH.DELETE

INTERNAL-USE METHODS:

cash.needs.input

comment	Method handling repetitive form-based input of budgeted cash requirements.
value	BORROWERS.MTH.CASH.NEEDS.INPUT

input

value	BORROWERS.MTH.INPUT

NAME:	FINANCIAL.MARKETS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	C$.MARKETS, FOREIGN.MARKETS
INSTANCES:	NIL
DESCRIPTION:	Definitional class for market objects, handling market condition parameters and future rate lists derived from event trees.

ATTRIBUTE SLOTS:

Lists of *current yield curve coefficients* over the planning period:
cf.current.a.coeffs
cf.current.b.coeffs

Lists of *current mean rates* over the planning period:
cf.mean.rates.lt
cf.mean.rates.st

Lists of *current rate changes* over the planning period:
cf.rate.changes.lt
cf.rate.changes.st

Default debt attribute values for use in creating new debts from borrowing plan actions:
default.call.first.premium
default.call.part?
default.call.wait.period
default.interest.pmts.per.year
default.issue.cost.rate

default.sf.contribution.rate
default.sf.wait.period

Constraint values for use in portfolio optimization, simulation or cash flow projection:

max.annual.borrowing
max.issue.size
min.annual.borrowing
min.issue.size

Parameters for pseudo-random rate generation:

quarter.alpha
quarter.sd.lt
quarter.sd.st
quarter.sd.random.part
year.alpha
year.sd.lt
year.sd.st
year.sd.st.random.part

Interest rate spreads for use in determining market rates for single financial instruments:

spread.bank.credit
spread.bank.deposit
spread.bond
spread.call
spread.prov.credit
spread.sf
spread.sf.earnings
spread.yield.to.call
term.lt
term.st

Other attributes:

currency

comment	Currency used in this market.
default	C$

issue.multiple

comment	Multiple amount in which debts are issued in this market.
default	10.0

rate.event.slot

comment	Name of slot in rate events containing rates for this market.

EXTERNAL-USE METHODS:

add

value	UNIT.MTH.ADD

change

value	UNIT.MTH.CHANGE

delete

value	UNIT.MTH.DELETE

generate.mean.rates

comment	Method to build mean-rate yield curve lists from mean-rate short-term and long-term rate lists, for use in optimization and cash flow projection.
value	MKT.MTH.GENERATE.MEAN.RATES

```
        generate.random.rates
              comment  Method to build pseudo-random yield curve
                       lists from mean-rate short-term and long-term
                       rate lists, for use in a single simulation pass.
              value    MKT.MTH.GENERATE.RANDOM.RATES
        interest.rate
              comment  Method to return the market interest rate for
                       a financial instrument in a sp6ecified time period.
              value    MKT.MTH.INTEREST.RATE
INTERNAL-USE METHODS:
        input
              value    MKT.MTH.INPUT
        interest.rate.model.input
              comment  Method for form-based input of rate model parameters.
              value    MKT.MTH.INTEREST.RATE.MODEL.INPUT
```

```
NAME:          C$.MARKET
SUBCLASS.OF:   FINANCIAL.MARKETS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Class for markets in domestic currency. All slots and methods
               inherited from financial.markets.
```

```
NAME:          FOREIGN.MARKETS
SUBCLASS.OF:   NIL
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Foreign market class (subclass of generic markets).
ATTRIBUTE SLOTS:
        cf.current.rates.fx
              comment  List of current exchange rates.
        cf.mean.rates.fx
              comment  List of mean exchange rates over the planning period,
                       based on the current rate event tree.
        cf.rate.changes.fx
              comment  List of intra-period exchange rate changes over the
                       planning period.
        fx.qtr.sd
              comment  Standard deviation of quarterly changes in exchange rates.
        fx.year.sd
              comment  standard deviation of yearly changes in exchange rates.
EXTERNAL-USE METHODS:
        fx.rate
              comment  Method to return the current exchange rate for a single
```

	financial instrument, based on the current rate list in an instance of this class.
value	FGN.MKT.MTH.FX.RATE

INTERNAL-USE METHODS:

fx.rate.model.input

comment	Method for form-based input of exchange rate model parameters.
value	FGN.MKT.MTH.FX.RATE.MODEL.INPUT

random.fx.rates

comment	Method to generate pseudo-random future exchange rates over the planning period, for use in a single simulation pass.
value	FGN.MKT.MTH.RANDOM.FX.RATES

NAME:	RATE.EVENTS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Rate events, linked by subclass relations into trees so that rates specified in an event are inherited unless changed.

ATTRIBUTE SLOTS:

event.branch

comment	Indicator that this event is the first in a new scenario. To be used in world creation and construction of optimization input.
range	T or NIL

event.description

comment	Description of event's causes or rationale.

event.period

comment	Index number of time period in which event occurs. Computed from event.year, event.quarter and planning period specifications as needed.

event.probability

comment	Conditional probability of event, given prior events in a rate scenario.

event.quarter

comment	Quarter in which event occurs.

event.year

comment	Year in which event occurs.

rates.canada

comment	New interest rates for Canada if this event takes place, in list of the form ((LT rate) (ST rate))

rates.us

comment	New interest and exchange rates for the US if this event occurs, in list of the form ((LT rate) (ST rate) (FX rate))

tree.name

comment	Name of the rate event tree containing this event.

EXTERNAL-USE METHODS:

```
    add
        value      RATE.EVENTS.MTH.ADD
    change
        value      RATE.EVENTS.MTH.CHANGE
    delete
        value      RATE.EVENTS.MTH.DELETE
    set.up.scenarios
        comment    Method to build mean-rate lists in financial markets at the
                   start of optimization, simulation or cash flow projection.
        value      RATE.EVENTS.MTH.SET.UP.SCENARIOS
INTERNAL-USE METHODS:
    input
        value      RATE.EVENTS.MTH.INPUT
    market.input
        comment    Method for form-based input of rates for a single market,
                   for a single event.
        value      RATE.EVENTS.MTH.MKT.INPUT
```

```
NAME:          DEBT.TYPES
SUBCLASS.OF:   NIL
INSTANCE.OF:   NIL
SUBCLASSES:    BOND.TYPES, ST.DEBT.TYPES
INSTANCES:     NIL
DESCRIPTION:   Top-level defining class for specifications for debt types from which
               an optimal portfolio is selected by the optimization model,
               or from which new debts are created from borrowing plans for simulation
               and cash flow projection.
ATTRIBUTE SLOTS:
    classes
        comment    Parent classes of the debt type.
    market
        comment    Market in which the debt type is issued.
        range      Markets defined in the current problem space.
    term
        comment    Term for the debt type.
    lp.name
        comment    Two-character name for this debt type used in the lp
                   column name according to the lp naming convention.
EXTERNAL-USE METHODS:
    create.plan
        comment    Top-level method controlling the creation of a hypothetical borrowing
                   plan action corresponding to a set of selected debt types.
    create.hypothetical.portfolio*
        comment    Top-level method controlling the creation of a hypothetical portfolio
                   based on a list of selected debt types, from which the
                   optimization model will select an optimal portfolio.
    select
        comment    Method to select a set of debt types to be considered by the
```

optimization model, from those defined for the problem space.

INTERNAL-USE METHODS:

create.actions		
	comment	Method to create hypothetical borrowing plan actions corresponding to a single debt type.
create.debts		
	comment	Method to create hypothetical debt objects for this debt type.
input		
	comment	Method for form-based input through which users define individual debt types.

NAME: BOND.TYPES
SUBCLASS.OF: DEBT.TYPES
INSTANCE.OF: NIL
SUBCLASSES: NIL
INSTANCES: NIL

DESCRIPTION: Defining class for bond types (specialization of debt types). Methods using the names defined in the parent class will be defined for this class.

ATTRIBUTE SLOTS:

call.wait		
	comment	Waiting period before bond type is callable, if type is a callable bond.
call.price		
	comment	First call price, if type is a callable bond.
classes		
	range	Any combination of bond classes defined in the main knowledge base.
sf.wait		
	comment	Waiting period before sinking fund contributions are required, if bond is a sinking fund bond.
sf.contribution.percent		
	comment	Sinking fund contribution rate, if bond is a sinking fund bond.

NAME: ST.DEBT.TYPES
SUBCLASS.OF: DEBT.TYPES
INSTANCE.OF: NIL
SUBCLASSES: NIL
INSTANCES: NIL

DESCRIPTION: Defining class for specifications for short-term debt types (specialization of debt types). Methods using the names defined in the parent class will be defined for this class.

ATTRIBUTE SLOTS:

classes		
	range	Short-term debt classes defined in the main knowledge base.
term		

```
          value      1
```

```
NAME:           BORROWING.ACTIONS
SUBCLASS.OF:    NIL
INSTANCE.OF:    NIL
SUBCLASSES:     BORROW.ACTIONS, OUTSTANDING.ACTIONS,
                RETIRE.ACTIONS, SURPLUS.ACTIONS, DELTA.ACTIONS
INSTANCES:      NIL
DESCRIPTION:    Definitional class for objects representing actions in a borrowing plan.
ATTRIBUTE SLOTS:
      action.date
            comment   Date on which action takes place, in Explorer
                      universal date format.
      action.period
            comment   Index of time period in which action occurs, based on
                      a planning period specification.
      action.quarter
            comment   Quarter in which action occurs. Computed from action date.
      action.source
            comment   Source for this action's specification.
            range     LP, USER or PF.SIMULATION
      action.year
            comment   Year in which the action takes place.
                      Computed from the action date.
      amount
            comment   Amount for this action. Will be 1 at start of optimization
                      and will be modified to equal decision variable values after optimization.
      debt.type
            comment   Debt type referred to by this action.
      fi.name
            comment   Name of the financial instrument associated with this action.
      issue.date
            comment   Date of issue for the financial instrument associated
                      with this action.
      issue.period
            comment   Index number of the time period of issue, computed
                      with respect to a planning period specification.
      issue.quarter
            comment   Quarter of financial instrument issue.
                      Computed from issue date.
      issue.year
            comment   Year of financial instrument issue.
                      Computed from issue date.
      lp.name
            comment   The name for the lp decision variable corresponding to this
                      action, according to the lp variable naming convention.
EXTERNAL-USE METHODS:
      add
```

```
        value      UNIT.MTH.ADD
    change
        value      UNIT.MTH.CHANGE
    create.debt
        comment    Method to create or modify the debt object corresponding to
                   a borrowing plan action.
    delete
        value      UNIT.MTH.DELETE
INTERNAL-USE METHODS:
    choose.type
        comment    Method used during input to select from a
                   list menu the action type to be created or modified.
        value      BP.MTH.CHOOSE.TYPE
    input
        value      BP.MTH.INPUT
```

```
NAME:          BORROW.ACTIONS
SUBCLASS.OF:   BORROWING.PLANS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for borrow actions. All slots and methods
               inherited from the parent class except for the 'create.debt'
               method, which is defined uniquely for these actions.
```

```
NAME:          OUTSTANDING.ACTIONS
SUBCLASS.OF:   BORROWING.PLANS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for outstanding actions. All slots and methods
               inherited from the parent class except for the 'create.debt'
               method, which is defined uniquely for these actions.
```

```
NAME:          RETIRE.ACTIONS
SUBCLASS.OF:   BORROWING.PLANS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Definitional class for retire actions. All slots and methods inherited
               from the parent class except for the 'create.debt' method,
               which is defined uniquely for these actions.
```

NAME:	SURPLUS.ACTIONS
SUBCLASS.OF:	BORROWING.PLANS
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Definitional class for surplus actions. All slots and methods inherited from the parent class except for the 'create.debt' method, which is defined uniquely for these actions.

NAME:	DELTA.ACTIONS
SUBCLASS.OF:	BORROWING.PLANS
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Definitional class for delta actions. All slots and methods inherited from the parent class except for the 'create.debt' method, which is defined uniquely for these actions.

4.1.3 LP Support objects

NAME:	LP.STRUCTURE.SPECIFIERS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	DECISION.VARIABLE.SPECIFIERS, CONSTRAINT.SPECIFIERS, OBJECTIVE.SPECIFIER
INSTANCES:	NIL
DESCRIPTION:	Parent class for objects that define the lp structure. Serves no purpose except to group objects by function in KEE graphic knowledge base displays.

NAME:	DECISION.VARIABLE.SPECIFIERS
SUBCLASS.OF:	LP.STRUCTURE.SPECIFIERS
INSTANCE.OF:	NIL
SUBCLASSES:	BORROW.DECISION.SPECIFIERS, OUTSTANDING.DECISION.SPECIFIERS, RETIRE.DECISION.SPECIFIERS, DELTA.DECISION.SPECIFIERS, SURPLUS.DECISION.SPECIFIERS
INSTANCES:	NIL
DESCRIPTION:	Parent class for decision variable specifiers, which build hypothetical borrowing decisions (actions) corresponding to

decision variables for optimization models.

ATTRIBUTE SLOTS:

decision.type

comment	One-character identifier for the decision type, used in the lp decision variable name.

decision.period.maximum

comment	Upper limit on decision periods used by an instance of this class to create separate actions.
range	(time period T, time period $T+1$, minimum of maturity period and T, minimum of maturity period and $T+1$, issue period only)

decision.period.values

comment	Decision period values, given an issue period value, for actions created by an instance of this class.
range	ISSUE.PERIOD, NEXT.PERIOD, FUTURE.PERIODS,

description.format

comment	Format string for the action description.

issue.period.source.slot

comment	Source slot in the debt object for the issue period value for an action created by an instance of this class. NIL indicates that the issue period is not used, as for delta decisions.
range	ISSUE.PERIOD, ALL.PERIODS, NIL

name.format

comment	Format string for decision variable name according to the LP naming convention.

object.classes

comment	Classes of object associated with actions created by an instance of this class.
range	LONG.TERM.DEBTS, SHORT.TERM.DEBTS, NIL

EXTERNAL-USE METHODS:

create.plan*

comment	Method controlling the creation of all action objects for all decision.variable.specifiers.

INTERNAL-USE METHODS:

create.actions*

comment	Method to create all actions for an instance of this class.

create.action*

comment	Method to create one action for an instance of this class.

NAME:	BORROW.DECISION.SPECIFIERS
SUBCLASS.OF:	DECISION.VARIABLE.SPECIFIERS
INSTANCE.OF:	NIL
SUBCLASSES:	BORROW.LT.DECISION.SPECIFIERS, BORROW.ST.DECISION.SPECIFIERS
INSTANCES:	NIL
DESCRIPTION:	Parent class for decision variable specifiers for borrow decisions.

ATTRIBUTE SLOTS:

decision.type

```
          value      B
     decision.period.maximum
          value      (time period T)
     description.format
          value      ('Borrow' + decision.amount + 'of' +
                     debt.type + 'in period' + decision.period.)
     name.format
          value      ('B' + 2-digit debt.type + 2-digit decision.period)
```

```
NAME:          BORROW.LT.DECISION.SPECIFIERS
SUBCLASS.OF:   BORROW.DECISION.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for decision variable specifiers for long-term
               borrowing decisions.
ATTRIBUTE SLOTS:
     decision.period.values
          value      ISSUE.PERIOD
     issue.period.source.slot
          value      ISSUE.PERIOD
     object.classes
          value      LONG.TERM.DEBTS
```

```
NAME:          BORROW.ST.DECISION.SPECIFIERS
SUBCLASS.OF:   BORROW.DECISION.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for decision variable specifiers for short-term
               borrowing decisions.
ATTRIBUTE SLOTS:
     decision.period.values
          value      ISSUE.PERIOD
     issue.period.source
          value      ALL.PERIODS
     object.classes
          value      SHORT.TERM.DEBTS
```

```
NAME:          OUTSTANDING.DECISION.SPECIFIERS
SUBCLASS.OF:   DECISION.VARIABLE.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    OS.LT.DECISION.SPECIFIERS,
               OS.ST.DECISION.SPECIFIERS
INSTANCES:     NIL
```

```
DESCRIPTION:  Parent class for decision variable specifiers for decisions to hold
              outstanding debt.
ATTRIBUTE SLOTS:
     decision.type
          value     O
     decision.period.maximum
          value     (minimum of maturity period and time period T+1)
     description.format
          value     ('Hold' + decision.amount + 'of' + debt.type
                    + 'issued in period' + issue period 'at the
                    beginning of period' + decision.period.)
     name.format
          value     ('O' + 2-digit debt.type + 2-digit
                    issue.period + 2-digit decision.period)
```

```
NAME:         OS.LT.DECISION.SPECIFIERS
SUBCLASS.OF:  OUTSTANDING.DECISION.SPECIFIERS
INSTANCE.OF:  NIL
SUBCLASSES:   NIL
INSTANCES:    NIL
DESCRIPTION:  Defining class for decision variable specifiers for decisions to
              hold long-term debt.
ATTRIBUTE SLOTS:
     decision.period.values
          value     FUTURE.PERIODS
     issue.period.source.slot
          value     ISSUE.PERIOD
     object.classes
          value     LONG.TERM.DEBTS
```

```
NAME:         OS.ST.DECISION.SPECIFIERS
SUBCLASS.OF:  OUTSTANDING.DECISION.SPECIFIERS
INSTANCE.OF:  NIL
SUBCLASSES:   NIL
INSTANCES:    NIL
DESCRIPTION:  Defining class for decision variable specifiers for decisions to
              hold short-term debt.
ATTRIBUTE SLOTS:
     decision.period.values
          value     NEXT.PERIOD
     issue.period.source
          value     ALL.PERIODS
     object.classes
          value     SHORT.TERM.DEBTS
```

```
NAME:          RETIRE.DECISION.SPECIFIERS
SUBCLASS.OF:   DECISION.VARIABLE.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    RETIRE.LT.DECISION.SPECIFIERS,
               RETIRE.ST.DECISION.SPECIFIERS
INSTANCES:     NIL
DESCRIPTION:   Parent class for decision variable specifiers for decisions to
               retire debt.
ATTRIBUTE SLOTS:
      decision.type
            value      R
      decision.period.maximum
            value      (minimum of maturity period and time period T + 1)
      description.format
            value      ('Retire' + decision.amount + 'of' +
                       debt.type + 'issued in period' + issue period
                       + 'in period' + decision.period.)
      name.format
            value      ('R' + 2-digit debt.type + 2-digit issue.period +
                       2-digit decision.period)
```

```
NAME:          RETIRE.LT.DECISION.SPECIFIERS
SUBCLASS.OF:   RETIRE.DECISION.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for decision variable specifiers for decisions
               to retire long-term debt.
ATTRIBUTE SLOTS:
      decision.period.values
            value      FUTURE.PERIODS
      issue.period.source.slot
            value      ISSUE.PERIOD
      object.classes
            value      LONG.TERM.DEBTS
```

```
NAME:          RETIRE.ST.DECISION.SPECIFIERS
SUBCLASS.OF:   RETIRE.DECISION.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Defining class for decision variable specifiers for decisions
               to hold short-term debt.
ATTRIBUTE SLOTS:
      decision.period.values
            value      NEXT.PERIOD
```

```
issue.period.source
     value      ALL.PERIODS
object.classes
     value      SHORT.TERM.DEBTS
```

```
NAME:          DELTA.DECISION.SPECIFIERS
SUBCLASS.OF:   DECISION.VARIABLE.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Parent class for decision variable specifiers for delta decisions
               handling integer values for minimum issue size constraints.
ATTRIBUTE SLOTS:
     decision.type
          value      D
     decision.period.maximum
          value      (time period T+1)
     description.format
          value      ('Delta decision of' + decision.amount + 'in period'
                     + decision.period)
     decision.period.values
          value      ALL.PERIODS
     issue.period.source.slot
          value      NIL
     name.format
          value      ('D' + 2-digit debt.type + 2-digit decision.period)
     object.classes
          value      NIL
```

```
NAME:          SURPLUS.DECISION.SPECIFIERS
SUBCLASS.OF:   DECISION.VARIABLE.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    NIL
INSTANCES:     NIL
DESCRIPTION:   Parent class for decision variable specifiers for decisions to hold
               a cash surplus balance.
ATTRIBUTE SLOTS:
     decision.type
          value      S
     decision.period.maximum
          value      (time period T+1)
     decision.period.values
          value      ALL.PERIODS
     description.format
          value      ('Hold a cash surplus of' + decision.amount +
                     'of' 'in period' + decision.period.)
```

```
issue.period.source.slot
      value       NIL
name.format
      value       ('S' + 2-digit decision.period)
object.classes
      value       NIL
```

```
NAME:          CONSTRAINT.SPECIFIERS
SUBCLASS.OF:   LP.STRUCTURE.SPECIFIERS
INSTANCE.OF:   NIL
SUBCLASSES:    CASH.REQUIREMENTS.CONSTRAINT.SPECIFIERS,
               INVENTORY.CONSTRAINT.SPECIFIERS.A,
               INVENTORY.CONSTRAINT.SPECIFIERS.B,
               MAX.DEBT.SVC.CONSTRAINT.SPECIFIERS,
               MAX.ISSUE.SIZE.CONSTRAINT.SPECIFIERS,
               MIN.ISSUE.SIZE.CONSTRAINT.SPECIFIERS,
               MATURITY.SMOOTHING.CONSTRAINT.SPECIFIERS,
               NONNEGATIVITY.CONSTRAINT.SPECIFIERS,
               BOUND.CONSTRAINT.SPECIFIERS,
               RANGE.CONSTRAINT.SPECIFIERS
INSTANCES:     NIL
DESCRIPTION:   Parent class for objects specifying the structure of individual
               lp constraints for use in building lp input files.
ATTRIBUTE SLOTS:
      constraint.type
            comment  2 character mnemonic constraint identifier, used to construct
                     constraint name according to lp naming convention.
      translation.format
            comment  Format string for text translation of meaning
                     of constraints specified by an instance of this class.
      decision.period.range
            comment  Range of decision periods over which constraints
                     of this type are defined.
            range    (1 through T, 2 through T + 1. ISSUE.PERIOD+1)
      issue.period.range
            comment  Range of issue periods over which constraints
                     of this type are defined.
            range    (0 through T − 2, 0 through T − 1)
      sum.over.debt.types?
            comment  Indicator as to whether or not constraints
                     specified by an instance of this class sum over k (debt type)
                     subscripts. If NIL, a separate constraint is created for each debt type.
            range    T, NIL
      sum.over.issue.periods?
            comment  Indicator as to whether or not constraints specified by
                     an instance of this class sum over s (issue period) subscripts.
                     If NIL, a separate constraint is created for each issue period
                     value. (If both sum indicators are NIL, a separate constraint
                     is created for each combination of the two.)
```

	range	T, NIL
borrow.decisions		
	comment	Indicator as to whether or not borrow decisions are used in constraints specified by an instance of this class.
	range	T, NIL
outstanding.decisions		
	comment	Indicator as to whether or not outstanding decisions are used in constraints specified by an instance of this class.
	range	T, NIL
retire.decisions		
	comment	Indicator as to whether or not retire decisions are used in constraints specified by an instance of this class.
	range	T, NIL
delta.decisions		
	comment	Indicator as to whether or not delta decisions are used in constraints specified by an instance of this class.
	range	T, NIL
surplus.decisions		
	comment	Indicator as to whether or not surplus decisions are used in constraints specified by an instance of this class.
	range	T, NIL

Each of the four slots immediately above has the following facets, which give decision and coefficient details for a type of decision variable in constraints specified by an instance of this class:

decision.period		
	comment	The t subscript for these decision variables in these constraints. Multiple values indicate that these decision variables are used more than once.
	range	t, $t-1$, $t+1$
coeff.source.objects		
	comment	Object class in which the coefficient is found for these decision variables in these constraints. Will have multiple values, in order corresponding to that of the values in the decision period range facet, if the decision variables are used more than once.
coeff.source.slots		
	comment	Slots in which the coefficients are found for these decision variables in these constraints. If more than one slot, values are totalled to give the coefficients.
coeff.source.index		
	comment	Specification for the time period index within cash flow lists for the coefficients for these decision variables in these constraints. NIL if the value is not taken from a list.
	range	CURRENT, PREVIOUS, NEXT
coeff.value		
	comment	Value of the coefficients for these decision variables in these constraints, if known rather than specified within an object in the knowledge base.
coeff.sign		
	comment	Sign for the coefficients of these decision variables

	in these constraints.
range	+, −
equation.type	
comment	Type of equation for these constraints.
range	*E* (indicating equal), *L* (indicating less than or equal), *G* (indicating greater than or equal), *N* (indicating nonnegativity).
rhs.source.object.class	
comment	Object class in which the right-hand side value is found for these constraints.
rhs.source.slot	
comment	Slot in which the right-hand side value is found for these constraints.
rhs.source.index	
comment	Specification for the time period index for the right-hand side value for these constraints. NIL is the value is not taken from a list.
rhs.value	
comment	Value of the constraint right-hand side if known rather than specified within an object in the knowledge base.
constraint.names	
comment	List of the names of constraints of this type for the current problem.

EXTERNAL-USE METHODS:

create.constraint.name.lists*	
comment	Method to control creation of lists of constraint names for the current problem and to store the names in the constraint specifier instances in the current problem space, and in the borrowing action object corresponding to the decision variable used in each type of constraint.
create.rows	
comment	Method to build input rows for this type of constraint. Called by input file builder methods.
sort.constraint.names.list	
comment	Method to sort constraint name lists into the order required for optimization model input.

INTERNAL-USE METHODS:

do.constraint.names	
comment	Method to create and store constraint names in a list in this unit.
do.constraint.variables	
comment	Method to create and store constraint names in decision variable (borrowing action) lists for use by input file builders.

NAME:	CASH.REQUIREMENTS.CONSTRAINT.SPECIFIERS
SUBCLASS.OF:	CONSTRAINT.SPECIFIERS
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Parent class for objects specifying the structure of cash

requirements constraints.

```
ATTRIBUTE SLOTS:
    constraint.type
        value    CR
    translation.format
        value    ('CR' + 2-digit decision.period)
    decision.period.range
        value    (1 through $T$)
    issue.period.range
        value    (0 through $T - 1$)
    sum.over.debt.types?
        value    $T$
    sum.over.issue.periods?
        value    $T$
    borrow.decisions
        value    $T$
    decision.period
        value    $t$
    coeff.source.objects
        value    CF.TABLES FOR DEBTS
    coeff.source.slots
        value    CF.PRINCIPAL.BORROWED, CF.ISSUE.COSTS
    coeff.source.index
        value    CURRENT
    coeff.sign
        value    +
    outstanding.decisions
        value    $T$
    decision.period
        value    $t$
    coeff.source.objects
        value    CF.TABLES FOR DEBTS
    coeff.source.slots
        value    CF.INTEREST.PAID, SF.CONTRIBUTIONS
    coeff.source.index
        value    CURRENT
    coeff.sign
        value    +
    retire.decisions
        value    $T$
    decision.period
        value    $t$
    coeff.source.objects
        value    CF.TABLES FOR DEBTS
    coeff.source.slots
        value    CF.RETIREMENT.COSTS, SF.WITHDRAWALS
    coeff.source.index
        value    CURRENT
    coeff.sign
        value    +
```

delta.decisions	
value	NIL
surplus.decisions	
value	T
decision.period	
value	t, $t+1$
coeff.source.objects	
value	
coeff.source.slots	
value	(CF.INV.BALANCE, CF.EARNINGS.RECEIVED), CF.INV.BALANCE
coeff.source.index	
value	CURRENT, NEXT
coeff.sign	
value	+
equation.type	
value	E
rhs.source.object	
value	PF.TABLE
rhs.source.slot	
value	CF.CASH.NEEDS
rhs.source.index	
value	CURRENT
rhs.value	
value	NIL

NAME:	INVENTORY.CONSTRAINT.SPECIFIERS.A
SUBCLASS.OF:	CONSTRAINT.SPECIFIERS
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Parent class for objects specifying the structure of inventory constraints for the period immediately after the issue period for a debt type.

ATTRIBUTE SLOTS:

constraint.type	
value	IV
translation.format	
value	('IV' + 2-digit debt.type + 2-digit issue.period + 2-digit decision.period)
decision.period.range	
value	(ISSUE.PERIOD+1)
issue.period.range	
value	(1 through T)
sum.over.debt.types?	
value	NIL
sum.over.issue.periods?	
value	NIL
borrow.decisions	
value	T

```
decision.period
        value      t - 1
coeff.source.objects
        value      NIL
coeff.source.slots
        value      NIL
coeff.source.index
        value      NIL
coeff.sign
        value      NIL
coeff.value
        value      -1
outstanding.decisions
        value      T
decision.period
        value      t
coeff.source.objects
        value      NIL
coeff.source.slots
        value      NIL
coeff.source.index
        value      NIL
coeff.sign
        value      NIL
coeff.value
        value      +1
retire.decisions
        value      NIL
delta.decisions
        value      NIL
surplus.decisions
        value      NIL
equation.type
        value      E
rhs.value
        value      0
```

Note: Other constraints are specified in a similar manner.

NAME:	INPUT.FILE.BUILDERS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	CORE.FILE.BUILDER, TIME.FILE.BUILDER, STOCH.FILE.BUILDER
DESCRIPTION:	Objects containing specifications and methods for building standard optimization input files.

ATTRIBUTE SLOTS:

contents
  comment List specifying the order of contents in the input file built by this object.
card.image.format
  comment Format string for ths FORTRAN card image format used for standard input lines.

EXTERNAL-USE METHODS:
build.file*
  comment Method to build file based on contents list.

NAME: CORE.FILE.BUILDERS
SUBCLASS.OF: NIL
INSTANCE.OF: NIL
SUBCLASSES: NIL
INSTANCES: CORE.FILE.BUILDER, TIME.FILE.BUILDER, STOCH.FILE.BUILDER
DESCRIPTION: Objects containing specifications and methods for building CORE file optimization input.
ATTRIBUTE SLOTS:
contents
  value ('NAME' 'ROWS' ROWS 'COLUMNS' COLUMNS 'RHS' 'BOUNDS' BOUNDS 'RANGES' RANGES 'ENDATA')

Note: Other input file builders are defined in a similar manner.

NAME: OUTPUT.TRANSLATOR
SUBCLASS.OF: NIL
INSTANCE.OF: NIL
SUBCLASSES: NIL
INSTANCES: NIL
DESCRIPTION: Object containing methods for translating optimization model output into knowledge base objects and slot values.
EXTERNAL-USE METHODS:
translate.output*
  comment Top-level methods controlling the sequence of output translation operations.
INTERNAL-USE METHODS:
record.decisions
  comment Method to store decision variable values in borrowing plan action objects and to delete unused actions and debt objects.
record.EVPI.data
  comment Method to store data on the expected value of perfect information in rate event objects, for later use in EVPI-based model modification. (Needed for branching scenario analysis rather than for the current system prototype.)

NAME:	COMMUNICATIONS.MANAGER
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Object containing methods for sending optimization input, requesting solution and receiving optimization output via DECNet.

ATTRIBUTES:
input.files.path.names
output.files.path.names
solver.login
solver.start.command

EXTERNAL-USE METHODS:
run.solver
comment Top-level control method for solver operation.

INTERNAL-USE METHODS:
invomslip
comment Method to contact the solver machine,
transmit input files and run the solver in background.
checkmslip
comment Method to check until the solver job is complete
and copy its output to the Explorer host.

4.2 System support subsystem

4.2.1 User interface objects

NAME:	I.O.UNITS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Definition for class of objects to handle menu and main window (MIDAS user interface screen) displays. One member is instantiated in each problem space.

ATTRIBUTE SLOTS:
bp.menu
comment Slot containing menu listing types of borrowing actions.
Used in borrowing plan modification.
display.menu
comment Slot containing menu listing display options following cash
flow simulation or projection. Menu is accessed by a

```
                    middle mouse click on the main window icon.
   list.menu
         comment    Slot holding a menu (which changes depending on current
                    activity) listing members of a specified object class. Used
                    by 'change' and 'delete' methods in knowledge base objects.
   midas.main.menu
         comment    Slot holding the main command menu for the system.
                    The menu is accessed by a left mouse click on the main
                    window icon.
EXTERNAL-USE METHODS:
   choose.from.list
         comment    Method to generate a list of object names and return
                    a user selection.
         value      I.O.UNIT.MTH.CHOOSE.FROM.LIST
   set.current.problem.space
         comment    Method to set up a knowledge base for the current
                    problem space if it does not already exist.
         value      I.O.UNIT.MTH.SET.CURRENT.PROBLEM.SPACE
   shutdown
         comment    Method to delete KEEPictures and the I.O.unit instance in the
                    current problem space prior to saving the problem space or
                    main knowledge base. Done to reduce storage requirements
                    and ensure a clean system startup without rebooting
                    the Explorer.
         value      I.O.UNIT.MTH.SHUTDOWN
   startup
         comment    Method to create all windows, menus and the main window
                    icon, set related global variables main a problem space
                    prior to system use.
         value      I.O.UNIT.MTH.STARTUP
INTERNAL-USE METHODS:
   delete.pictures
         comment    Method to delete all KEEPictures. Used by the shutdown
                    method.
         value      I.O.UNIT.MTH.DELETE.PICTURES
   msg.window
         comment    Method to expose a window and send a message for which
                    a reply is not expected.
         value      I.O.UNIT.MTH.MSG.WINDOW
   print.slot.value
         comment    Utility method to print a specified slot value on request
                    from the main menu.
         value      I.O.UNIT.MTH.PRINT.SLOT.VALUE
   prompt.window
         comment    Method to expose a window, send a user prompt and return
                    the reply.
         value      I.O.UNIT.MTH.PROMPT.WINDOW
   shrink.viewports
         comment    Utility method to shrink the main window to a bar, enabling
                    a return to the development system without a
```

	full-scale shutdown and restart.
value	I.O.UNIT.MTH.SHRINK.VIEWPORTS

Note: Specifications for bitmaps, the icon, windows. menus and miscelleanous display functions are found in the files MIDASPT.LISP and UI-METHODS.LISP on microfiche.

4.2.2 Output managers

NAME:	CF.TABLES
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	FI.TABLES, PF.TABLES
INSTANCES:	NIL
DESCRIPTION:	Generic class defining tables of cash flow lists, along with general methods and facet parameters to do standard computations.

ATTRIBUTE SLOTS:

Cash flow detail slots, containing lists of cash flow detail amounts by time period:

cf.interest.paid
cf.issue.costs
cf.principal.borrowed
cf.principal.invested
cf.principal.retired
cf.principal.withdrawn
cf.retirement.costs
sf.contributions
sf.earnings
sf.withdrawals

Memo slots, containing lists of non-cash flow items by time period:

cf.interest.earned
cf.earnings.accrued
cf.earnings.rate
cf.fx.rate
cf.interest.accrued
cf.interest.rate
cf.inv.balance
cf.net.principal.outstanding
cf.principal.outstanding
cf.remaining.term
cf.status
sf.balance
sf.earnings.accrued
sf.earnings.rate
sf.status

Cash flow and memo slot facets, attached to each such slot and used by table methods:

active

comment	Indicator as to whether the slot is used for cash flows by the source object.

initial.element

comment	Symbol to be used to initialize the slot's list at the start of optimization, simulation or cash flow projection. Value depends on the type of data in the list.
range	0, 0.0 or NIL

title

comment	Line item title string for use in displays of the slot's contents.

Total slots, containing lists of total amounts by time period:

cf.total.cash.flows
cf.total.earnings
cf.total.operating.cash.flows
sf.total.cash.flows
sf.total.earnings

Total slot facets, attached to each total slot and used by table methods:

active

comment	Indicator as to whether the slot is used for cash flows by the source object.

initial.element

comment	Symbol to be used to initialize the slot's list at the start of optimization, simulation or cash flow projection. Value depends on the type of data in the list.
range	0, 0.0 or NIL

sum.of

comment	List of names of slots to be totalled by time period to give the value of this slot.

title

comment	Line item title string for use in displays of the slot's contents.

Net slots, containing lists of net amounts by time period:

cf.net.cash.flows
cf.net.operating.cash.flows

Net slot facets, attached to each net slot and used by table methods:

active

comment	Indicator as to whether the slot is used for cash flows by the source object.

initial.element

comment	Symbol to be used to initialize the slot's list at the start of optimization, simulation or cash flow projection. Value depends on the type of data in the list.
range	0, 0.0 or NIL

net.of

comment	Names of the slots for which the difference by time

		period gives the value of this slot.
title		
	comment	Line item title string for use in displays of the slot's contents.

NPV slots, containing the discounted value of lists elsewhere in the table:

npv.total.cash.flows

NPV slot facets:

active		
	comment	Indicator as to whether the slot is used for cash flows by the source object.
base.slot		
	comment	Name of the slot to be discounted to give this slot's value.
initial.element		
	comment	Symbol to be used to initialize the slot at the start of optimization, simulation or cash flow projection.
	value	0.0
title		
	comment	Line item title string for use in displays of the slot's contents.

IRR slots, containing the internal rate of return (if defined) for lists elsewhere in the table:

irr.total.cash.flows

IRR slot facets:

active		
	comment	Indicator as to whether the slot is used for cash flows by the source object.
base.slot		
	comment	Name of the slot to be used to compute this slot's value.
initial.element		
	comment	Symbol to be used to initialize the slot at the start of optimization, simulation or cash flow projection.
	value	0.0
title		
	comment	Line item title string for use in displays of the slot's contents.

Simulation result slots, containing lists of results from multiple simulation passes:

sim.end.value
sim.irr.total.cash.flows
sim.npv.total.cash.flows

Simulation result slot facets, attached to each such slot and containing the results of the statistical analysis of the slot's values:

base.slot		
	comment	Name of the slot to be used to compute this slot's value.
high		
	comment	Maximum element in the list of results.
low		
	comment	Minimum element in the list of results.
mean		
	comment	Mean result value.
quartile.1		

```
        comment   First-quartile value in the list of results.
    quartile.2
        comment   Second-quartile value in the list of results.
    quartile.3
        comment   Third-quartile value in the list of results.
    std.dev
        comment   Standard deviation of the results.
    title
        comment   Line item title string for use in displays of the slot's contents.
```

Display slots, filled in and used by display methods, containing formatted titles and data for table displays of data in the table:

```
    cf.display.data
    cf.display.titles
    earnings.display.data
    earnings.display.titles
    ocf.display.data
    ocf.display.titles
    sf.cf.display.data
    sf.cf.display.titles
    sf.earnings.display.data
    sf.earnings.display.titles
```

Slot name lists, containing lists of slot names for use by table methods:

```
    cf.slots
    fx.slots
    irr.slots
    net.slots
    npv.slots
    sim.result.slots
    total.slots
```

Source slots, documenting the source of data in the table:

```
    source.class
    source.name
```

EXTERNAL-USE METHODS:

```
    analyze.sim.results
        comment   Method to do statistical analysis and fill in
                  facet values for simulation result slots.
        value     CF.TABLES.MTH.ANALYZE.SIM.RESULTS
    initialize
        comment   Method to initialize slots in a table prior
                  to cash flow calculations.
        value     CT.TABLES.MTH.INITIALIZE
    initialize.sim.results
        comment   Method to initalize simulation result slots.
        value     CF.TABLES.MTH.INITIALIZE.SIM.RESULTS
    nets
        comment   Method to compute net lists for specified slots.
        value     CF.TABLES.MTH.NETS
    npvs
        comment   Method to compute net present values for specified slots.
```

	value	CF.TABLES.MTH.NPVS
setup.cf.table		
	comment	Method to create a new table for a model object and store the required cross-reference pointers between the table and the source object.
	value	CF.TABLES.MTH.SETUP.CF.TABLE
store.sim.results		
	comment	Method to store simulation results in lists.
	value	CF.TABLES.MTH.STORE.SIM.RESULTS
totals		
	comment	Method to compute total values for specified lists.
	value	CF.TABLES.MTH.TOTALS
translate.fx		
	comment	Method to translate specified slots to their Canadian dollar equivalents.
	value	CF.TABLES.MTH.TRANSLATE.FX

NAME:	FI.TABLES
SUBCLASS.OF:	CF.TABLES
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Class for cf.tables for individual financial instruments. All slots and methods inherited from parent class.

NAME:	PF.TABLES
SUBCLASS.OF:	CF.TABLES
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Extension of cf.tables for portfolios; contain performance indicators, constraint comparisons and cash surplus/deficit data.

ATTRIBUTE SLOTS:

Slots used for processing *cash requirements* and the *cash deficit or surplus*:

cf.cash.deficit
cf.cash.needs

Each of these slots has *active, initial.element and title* facets as described in CF.TABLES, above.

Performance indicator slots:

cf.average.cost
cf.average.term
cf.percent.foreign
cf.percent.lt
cf.percent.mt
cf.percent.st
cf.percent.variable

Performance indicator slot facets:

condition		
	comment	Condition to be satisfied if a portfolio member is to be included in this indicator.
denominator.slot		
	comment	Slot in portfolio member where a value is found that should be added to the indicator's denominator.
multiplier.slot		
	comment	Slot in portfolio member where a value is found with which to multiply the numerator value.
numerator.slot		
	comment	Slot in portfolio member where a value is found to be added to the indicator's numerator.
percent		
	comment	Indicator as to whether or not this indicator is a percentage and should be multplied by 100.

Each of these slots also has *active, initial.element and title* facets as described in CF.TABLES, above.

Constraint comparison slots:

cf.max.debt.svc
cf.max.debt.retirement
cf.min.debt.retirement

Constraint comparison slots facets:

base.slot		
	comment	Slot in table containing list to be compared against constraint.
source		
	comment	Name of object containing constraint value.
source.slot		
	comment	Name of slot in source object containing constraint value.

Each of these slots also has *active, initial.element and title* facets as described in CF.TABLES, above.

Slot name lists:

cf.slots
line.item.slots
constraint.slots
pi.slots

Each of these slots has *active, initial.element and title* facets as described in CF.TABLES, above.

Display slots:

pi.display.data
pi.display.titles

EXTERNAL-USE METHODS:

check.constraints		
	comment	Method to compare constraints against result lists.
line.totals		
	comment	Method to compute line-by-line totals of member results to give portfolio results.
	value	PF.TABLES.MTH.LINE.TOTALS

```
        perf.indicators
            comment  Method to calculate performance indicators for pi.slots.
            value    PF.TABLES.MTH.PERF.INDICATORS
```

```
NAME:          PRESENTATION.MANAGERS
SUBCLASS.OF:   NIL
INSTANCE.OF:   NIL
SUBCLASSES:    UNIT.DISPLAY.MANAGERS, TEXT.DISPLAY.MANAGERS,
               SLOT.DISPLAY.MANAGERS, GRAPH.DISPLAY.MANAGERS,
               HEADING.DISPLAY.MANAGERS, TABLE.DISPLAY.MANAGERS
INSTANCES:     NIL
DESCRIPTION:   Generic class for objects to produce output displays and graphs.
ATTRIBUTE SLOTS:
    contents
        comment  List specifying display contents.
    data.repaint.function
        comment  Name of function to display contents of data (scrolling)
                 window for this display.
    data.slot
        comment  Name of slot in the table to be displayed in which
                 data contents will be stored after formatting.
    data.window
        comment  Name of data window to be used by this display.
    output.type
        comment  Name of display, to be used in formatting title.
    title.repaint.function
        comment  Name of function to display contents of title (nonscrolling)
                 window for this display.
    title.slot
        comment  Name of slot in the table to be displayed in which
                 title contents will be stored after formatting.
    title.string
        comment  Format string for display title.
    title.window
        comment  Name of title window to be used for this display.
EXTERNAL-USE METHODS:
    display
        comment  Control method to format and display this output.
INTERNAL-USE METHODS:
    format.title
        comment  Method to format the display title.
    setup
        comment  Method to initialize window streams and function names
                 prior to the actual display.
```

Note: Specifications for output manager subclasses, instances and methods are found in the files MIDASPT.U and UI-METHODS.LISP on microfiche.

NAME:	LP.HISTORY.OBJECTS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBLCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Definitional class for objects holding the history of lp model runs and modifications. A LP history object is instantiated for each run of the lp in a problem space.

ATTRIBUTE SLOTS:

run.id

comment	Identifier for lp run.
default	INITIAL

run.date.and.time

comment	Date and time of this run, in Explorer universal date format.

slot.changes

comment	List of slot changes prior to this run, in list of form ((object.name slot.name old.value new.value changed.by)...)

object.changes

comment	List of object changes prior to this run, in list of form ((object.name change.type changed.by)...)

rate.event.tree.changes

comment	List of rate event tree changes prior to this run, in list of form ((node.name change.type changed.by)...)

Methods for this class will be defined as system development progresses.

4.2.3 Task managers

NAME:	TASKS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	SUBCLASSES DEFINING SPECIFIC TASKS.
INSTANCES:	
DESCRIPTION:	Defining class for tasks for use in system control. Specific task objects are instantiated in the problem space.

ATTRIBUTE SLOTS:

conditions

comment	Names of condition objects representing conditions to be satisfied before this tasks can be executed.

subtasks

comment	Names of subtasks of this task.

next.task

	comment	Name of task which is normally performed following this task.
alternate.tasks		
	comment	Names of tasks which may be performed if the user decides not to perform this task.
long.prompt		
	comment	Prompt which explains this task and asks for user confirmation.
short.prompt		
	comment	Prompt which briefly asks for confirmation of this task, used as an alternative to the long.prompt based on a flag set by the user.
why.perform?		
	comment	String describing the task, for use in task explanations.
confirm?		
	comment	Indicator as to whether or not this task requires user confirmation before execution.
	range	T, NIL
required?		
	comment	Indicator as to whether or not this task must be performed for a given analysis type.
	range	T, analysis type names
schedule.status		
	comment	Indicator as to whether or not this task has been placed on the system agenda.
	range	T, NIL
condition.status		
	comment	Indicator as to whether or not all conditions for this task are satisfied.
	range	T, NIL
confirmed.status		
	comment	Indicator as to whether or not this task has been confirmed by the user.
	range	T, NIL
performed.status		
	comment	Indicator as to whether or not this task has been performed.
	range	T, NIL
subtask.status		
	comment	Indicator as to whether or not all subtasks of this task have been performed.
	range	T, NIL

EXTERNAL-USE METHODS:

perform		
	comment	Method to start execution of this task, by either starting rule-based reasoning or sending a message.

INTERNAL-USE METHODS:

confirm		
	comment	Method to request and process user confirmation or rejection of this task.
check.conditions		

comment	Method controlling condition checks for this task.

NAME:	TASK.CONTROLLERS
SUBCLASS.OF:	NIL
INSTANCE.OF:	NIL
SUBCLASSES:	NIL
INSTANCES:	NIL
DESCRIPTION:	Defining class for task control object, which maintains agenda and starts task control reasoning.

ATTRIBUTE SLOTS:

task.agenda		
	comment	List of tasks scheduled for consideration, in order of planned performance. The current task is the first element in the list.
current.task		
	comment	Task currently under consideration for performance.
analysis.type		
	comment	Name of analysis being performed.
	range	OPTIMIZE.PLAN, REFINE.PLAN, SIMULATE.PLAN, SIMULATE.PF, SIMULATE.DEBT, PROJECT.PLAN, PROJECT.PF, PROJECT.DEBT.
completed.tasks		
	comment	List of names of completed task for this analysis session. Used to store session status in problem space file.
	range	Task names
conditions.ok.tasks		
	comment	List of names of tasks for which conditions are satisfied for this analysis session. Used to store session status in problem space file.
	range	Task names
confirmed.tasks		
	comment	List of names of tasks which have been confirmed by the user during this analysis session. Used to store session status in problem space file.
	range	Task names
end.status		
	comment	Indicator as to whether session is to end before a new task is started. Set by task control rules at end of analysis step.
	range	T, NIL
prompt.type		
	comment	Type of prompt to be shown to user.
	range	LONG, SHORT

EXTERNAL-USE METHODS:

begin.assisted.analysis		
	comment	Method initiating rule-based reasoning for system control.

```
NAME:         TASK.CONDITIONS
SUBCLASS.OF:  NIL
INSTANCE.OF:  NIL
SUBCLASSES:   SPECIFIC TASK CONDITION CLASSES.
INSTANCES:    NIL
DESCRIPTION:  Defining class for task conditions, used by tasks in task control.
ATTRIBUTE SLOTS:
      condition.status
           comment  Indicator as to whether or not this condition is currently
                    satisfied in the knowledge base.
      condition.description
           comment  Description of this condition, for use in long.prompt
                    and in explanations.
      long.prompt
           comment  Explanatory prompt for use when the user must confirm
                    that a condition is satisfied.
      short.prompt
           comment  Short prompt for use when the user must confirm that a
                    condition is satisfied. Used rather than the long.prompt if
                    a user flag is set.
EXTERNAL-USE METHODS:
      check.condition
           comment  Method to check whether this condition is satisfied,
                    either by initiating rule-based reasoning or sending a message.
      confirm.condition
           comment  Method to confirm with the user than this condition is satisfied.
```

4.3 User support subsystem

```
NAME:         RESULT.ANALYZERS
SUBCLASS.OF:  NIL
INSTANCE.OF:  NIL
SUBCLASSES:   LP.ANALYZERS, SENSITIVITY.ANALYZERS
INSTANCES:    NIL
DESCRIPTION:  Defining class for objects guiding analysis of model results.
              Subclasses are defined in a hierarchy, ending in a
              specific analyzer for each type of result to be analyzed.
ATTRIBUTE SLOTS:
      result.type
           comment  Identifier for the type of result analyzed by
                    an instance of this class.
      result.source
           comment  Name of object containing results to be analyzed.
      result.description
           comment  Description of the result type, for use in explanations.
```

```
        possible.key.factors
              comment  List of key factors to be considered as possible causes or
                       contributing factors for the results being analyzed.
        actual.key.factors
              comment  List of key factors identified as actual causes or
                       contributing factors for the results being analyzed.
        analysis.tasks
              comment  List of tasks to be carried out for this type of analysis
                       once key factors are identified.
EXTERNAL-USE METHODS:
        check.key.factors
              comment  Method controlling key factor checking for a set of
                       results by an instance of this class.
        schedule.analysis.tasks
              comment  Method to schedule the analysis tasks for this analysis
                       once key factors are identified.
```

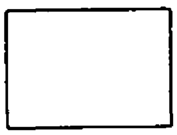
Note: The subclasses of RESULT.ANALYZERS are identified in Figure 8.2. They will be further defined as result analysis requirements become clear during further work.

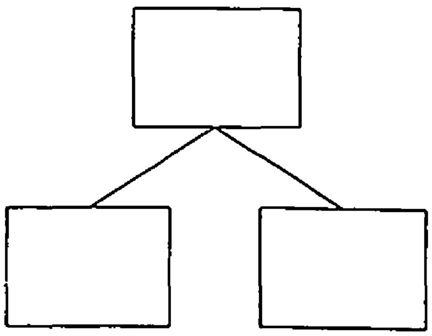
```
NAME:           KEY.RESULT.FACTORS
SUBCLASS.OF:    NIL
INSTANCE.OF:    NIL
SUBCLASSES:     SUBCLASSES FOR SPECIFIC KEY FACTORS, SUCH AS
                LARGEST.DEBT, HIGHEST.END.VALUE.DEBT, ETC.
INSTANCES:      NIL
DESCRIPTION:    Defining class for objects representing possible key factors
                to be investigated during result analysis.
ATTRIBUTE SLOTS:
        result.types
              comment  Types of results for which an instance of this class
                       is a possible key factor.
        result.source
              comment  Object containing the results being analyzed.
        key.factor.status
              comment  Indicator as to whether or not this factor is a cause
                       or contributing factor in the results being analyzed.
EXTERNAL-USE METHODS:
        check.factor
              comment  Method to check for this factor in a set of results.
```

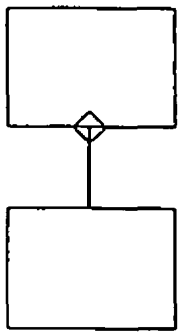
---

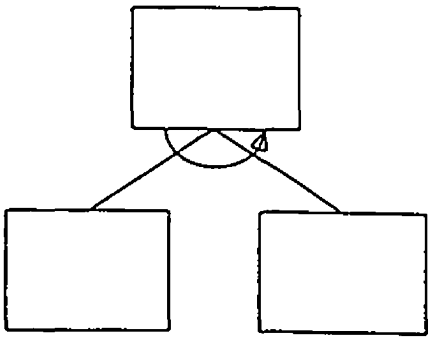
5. Method structure charts

Structure charts are used to document the control structures of significant high-level methods within MIDAS. These charts use the following symbols and conventions, which are simplifications of the standard notation documented in Whitten *et al.* (1989):

1. 
A rectangle represents a program (method) module, which may be either a separate method or a subsection of a method. Each module carries out a single well-defined procedure. The object in which the module is located within the system is used as the first part of the module label.

2. 
A solid line represents a module relationship in which the upper module calls and passes parameters to the lower module. Control and data parameters are not shown on these charts but can be seen in the code listings on microfiche.

3. 
A diamond indicates the conditional execution of one or more submodules based on conditions noted in the chart.

4. 
An arc-shaped arrow is a 'loop' symbol, indicating repetition of one or more modules based on conditions noted in the chart.

Method structure charts in this documentation are the following:

1. Portfolios: Optimize: Figure 6.20.
2. Portfolios: Configure.optimization.model: Figure 6.21.
3. Portfolios: Simulate: Figure 6.22.
4. Financial.instruments: Do.cash.flows: Figure 6.23.
5. Table.display.managers: Display: Figure 7.10.
6. Parametric analysis: Figure 8.5.

6. Detailed method specifications

Detailed method specifications are given in Structured English with the following conventions: (a) submodules, which may be either separate methods or parts of the method being specified, are indented; (b) Slot values are shown in the form OBJECT: SLOT.NAME; (c) method calls are shown in the form OBJECT: METHOD.NAME; and (d) THISUNIT refers to the object in which the method is called.

Methods which are working in the system prototype have been documented through comments in the function files on microfiche. Specifications for significant methods which are not yet implemented are given here by subsystem and object class, in the order of the object definitions given earlier, for single-scenario models. Extension to branching scenarios and worlds would require loops for each world, world specification for slot value changes, additional summary objects and methods and the use of the TIME and STOCH files for LP input.

6.1 Modelling subsystem methods

6.1.1 Model object methods

```
Portfolios: Optimize
    Initialize.optimization
        Confirm problem description with user
        For each market:
            Market: Generate.mean.rates
        Debt types: Create.hypothetical.portfolio
    Hypothetical.portfolio: Configure.optimization.model
    Communications.manager: Run.solver
    Terminate optimization
        Output.translator: Translate.output
        Main.heading.display.managers: Display
        LP.results.display.manager: Display
```

```
Portfolios: Configure.optimization.model
    Build LP structure
        Decision.variable.specifiers: Create.plan
        Constraint.specifiers: Create.constraint.name.lists
        Objective.specifier: Create.objective.name.list
    Do LP coefficients
        Hypothetical.portfolio: Do.cash.flows
    Build input files
        Core.file.builder: Build.file
```

6.1.2 Model support object methods

```
Debt.types: Create.hypothetical.portfolio
    Format the hypothetical portfolio name
    If the hypothetical portfolio does not exist
        Create it as a child of PORTFOLIOS
    Else initialize its members list to NIL
    Set the portfolio members list equal to members of existing
    debt portfolio
    For each debt type
        Debt.type: Create.debts
    If the object CASH.SURPLUS does not exist
        Create it as a child of SHORT.TERM.INVESTMENTS
        Add its name to the list of portfolio members
    If the object CASH.DEFICIT does not exist
        Create it as a child of PENALTY.BORROWING
        Add its name to the list of portfolio members
    Store the hypothetical portfolio name as the name of the
    current portfolio
```

```
Debt.types: Create.debts
    For each time period in planning period
        Format a debt name
        Set parent.classes equal to Debt.type: classes
        If debt object of that name does not exist, create it
        with specified parent classes
```

```
Store object attributes
    Set debt: LP name to debt.type: LP name
    Set debt: market equal to debt.type: market
    Set debt: term equal to debt.type: term
    If not a foreign debt, set debt:
    original.principal and debt: principal.outstanding
    equal to 1.0
    Else set debt: original.principal and debt:
    principal.outstanding equal to $1/fx rate for the
    time period. (**don't have rates yet)
    Set other debt parameters, depending on debt type,
    from defaults in debt.type: market
    Add debt name to portfolio members list
```

6.1.3 LP support object methods

```
Decision.variable.specifiers: Create.plan
    For each subclass of decision.variable.specifiers
        Subclass: Create.actions
```

```
Decision.variable.specifiers: Create.actions
    Set decision.amount equal to 0.0
    For each debt named in thisunit's object.classes slot which
    is also in the hypothetical portfolio
        Set debt.type equal to debt: lp.name
        Set issue.period equal to the value in the debt object
        of the slot specified in the issue.period.source.slot
        Set decision.period.values equal to thisunit:
        decision.period.values
        Set decision.period.maximum equal to thisunit:
        decision.period.maximum
        If decision.period.values is equal to ISSUE.PERIOD
            Set decision.period equal to issue.period
            If the decision.period is less than the
            decision.period.maximum
                Thisunit: create.action
        Else if decision.period.values is equal to NEXT.PERIOD
            Set decision.period equal to issue.period+1
            If the decision.period is less than the
            decision.period.maximum
                Thisunit: create.action
        Else if decision.period.values is equal to
        FUTURE.PERIODS
            For each time period from issue.period to end of
            planning period
                Set decision.period equal to the index of the
                time period
                If the decision.period is less than the
                decision.period.maximum
                    Thisunit: create.action
        Else if decision.period.values is equal to ALL.PERIODS
            For each time period from the beginning to the end
            of planning period
                Set decision.period equal to the index of the
                time period
                If the decision.period is less than the
                decision.period.maximum
                    Thisunit: create.action
```

```
Decision.variable.specifiers: Create.action
    Format the decision.variable.name using the name format in
    thisunit and the issue.period and decision.period set by the
    create.actions method
    If an action of the decision.variable.name does not exist,
    create it in the problem space as an instance of the class
    named in thisunit: decision.parent.class
    Else set its amount slot value to 0.0
    Set action: action.period to decision.period
    Set action: action.year to the year corresponding to
    action.period
    Set action: action.quarter to the quarter corresponding to
    action.quarter
    Set action: debt.type to debt.type
    Set action: fi.name to the name of the debt for which this
    action was created
    Set action: issue.period to issue.period
    Set action: issue.year to the year corresponding to
    issue.period
    Set action: issue.quarter to the quarter corresponding to
    issue.period
    Set action: lp.name to lp.name
```

```
Constraint.specifiers: Create.constraint.name.lists
    For each subclass of Constraint.specifiers
        Create a subclass instance in the problem space
            Instance: Do.constraint.names
            Instance: Do.constraint.variables
            Instance: Sort.constraint.name.list
        Create a master list of all constraint names, sort and
        store in the current lp history object
    For each instance of borrowing.plans
        Instance: Sort.constraint.name.list
```

```
Constraint.specifiers: Do.constraint.names
    Set constraint.names list in thisunit to NIL.
Set constraint.type to value of thisunit: constraint.type
Do for $t$ from beginning to end of thisunit:
decision.period.range
    If thisunit: sum.over.debt.types? is equal to $T$
        Do for $k$ equals 1
            If thisunit: sum.over.issue.periods? is equal to $T$
                Do for $s = 1$
                Else do for $s$ from beginning to end of
                thisunit: issue.period.range
                    Format a constraint name using
                    constraint.type, $k$, $s$ and $t$
                    Add the constraint name to
                    thisunit: constraint.names
            Else do for each selected debt type and each
            existing debt
                Set $k$ equal to the lp.name of the debt
                type or existing debt
                If thisunit: sum.over.issue.periods? is
                equal to $T$
                    Do for $s = 1$
                        Format a constraint name using
                        constraint.type, $k$, $s$ and $t$
                        Add the constraint name to
                        thisunit: constraint.names
                Else do for $s$ from beginning to end of
                thisunit: issue.period.range
                    Format a constraint name using
                    constraint.type, $k$, $s$ and $t$
                    Add the constrainc name to
                    thisunit: constraint.names
```

```
Constraint.specifiers: Do.constraint.variables
    For each constraint name in thisunit: constraint.names
        Use the specifications in thisunit to build the names
        of the decision varibles used in the constraint
        For each such decision variable, add the constraint
        name to the constraint.names slot in the corresponding
        borrowing.action.
```

Note: This method uses the same generalized approach as constraint.specifiers: Do.constraint.names.

```
Core.file.builder: Build.file
    Name and open core file
    Format and write the NAME record
    Format and write a ROWS record
    Format and write an objective name record
    For each constraint name in the master list in the lp
    history object
```

```
            Format and write a ROW record
      Format and write a COLUMNS record
      For each borrowing plan action
            For each constraint name in action: constraint.names
                  Use the coefficient specifications in the
                  corresponding constraint specifier to determine
                  the action's coefficient
                  Format and write a COLUMN record with the
                  coefficient value
      Format and write a RHS record
      For each constraint name in the master list in the lp
      history object
            Format and write a RHS record
      Do BOUNDS header and records using the same approach
      Do RANGES header and records using the same approach
      Format and write an ENDDATA record
```

Note: Other file builders operate in the same manner.

```
Output.translator: Translate.output
      Do until end of file:
            Read an lp output file record
            Set card.id equal to the first field in the record
            If card.id equals NAME
                  Set problem.name equal to the second field in the record
                  Check that problem.name equals Core.file.builder:
                  problem.name and Time.file.builder: problem.name
            Else if card.id equals STATUS
                  Set status equal to the second field in the record
                  If status not equal SOLVED return status and end
            Else if card.id equals TIME
                  Set time.period.index equal to the second field in
                  the record
            Else if card.id equals VALUE
                  Set the objective value in the current LP.history
                  object equal to the second field in the record
            Else if card.id equals VARIABLES
                  Do until DUALS card is reached
                        Read a new record
                        Set variable.name equal to the first field in
                        the record
                        Set action.name equal to the name of the
                        borrowing action having variable.name as its
                        lp.name slot value
                        Set the AMOUNT field in the action.name
                        object equal to the second field in the record
      Borrowing.actions: Create.actual.portfolio
```

Note: This definition handles only output for single-scenario problems.

Branching scenario problems have SCENARIO header cards, which mark the beginnings of new scenarios and worlds.

6.2 System support subsystem

6.2.1 Output management methods

All output management methods are documented in the files CFT-METHODS.LISP and UI-METHODS.LISP on microfiche, with the exception of **Portfolios: Check constraints**, which uses the same technique as the others to compare the constraint value period-by-period against list elements in the pf.table's base.slot value.

6.2.2 Task management methods

Top-level task management methods consist of single-line requests to start rule-based reasoning. Individual task performance, confirmation and condition checking methods are straightforward procedures and are individually defined by task or condition.

6.3 User support subsystem

Methods for the user support subsystem must be defined specifically for each result type and key factor type and will be defined through further work. A structure chart illustrating a typical user support method is given in Figure 8.5.

7. Rule specifications

The rules used in the MIDAS prototype are given in the following figures in the text of this dissertation:

1. Heuristic plan refinement rules: Figures 6.18 and 6.19.
2. Task control rules: Figure 7.16.

Appendix D

Prototype System Code: Comments

LISP code listings for the MIDAS prototype system, found on pages D1–D184 of the microfiche inside the back cover of this volume, contain (a) the main knowledge based definition, generated by KEE, (b) method definitions which are accessed through messages to method slots in KEE objects, (c) global variable, window and bitmap definitions, and (d) a few miscellaneous functions which are used in methods but were defined as LISP functions.

File contents are as follows:

1. CFT-METHODS: Methods for cash flow tables.
2. DEBT-METHODS.LISP: Methods for all classes of bonds and short-term debts.
3. INV-METHODS.LISP: Methods for long-term investments (sinking funds) and short-term investments (cash surplus).
4. MIDASPT.LISP: Bitmaps for the user interface, generated by KEE.
5. MIDASPT.U: KEE knowledge base definition.
6. MISC-FUNCTIONS: Non-method functions; also methods, including UNIT.ADD, CHANGE and DELETE, used by more than one object class.
7. MKT-METHODS.LISP: Methods for financial markets and rate event trees.
8. MO-METHODS.LISP: Methods for model objects not found elsewhere (portfolios and financial instruments).
9. MS-METHODS.LISP: Methods for model support objects not found elsewhere (borrowers, problem specifiers, debt types, borrowing plans).

10. UI-METHODS.LISP: Methods, window definitions, menu definitions and global variable settings for the user interface.

    Assistance in understanding LISP syntax and functions can be found in Steele (1984).

Appendix E

Test Data: Comments

Microfiche pages E1–E100 list the KEE-generated problem space file containing a comprehensive set of test data for the MIDAS prototype. A subset of this data produced the examples in Chapter 10. Rates and rate model parameters used are hypothetical and are used merely to test and illustrate the system's operation.

References

Adamidou, E., Ben-Dov, Y., and Pendergast, L. (1989). Optimal structured portfolios under various interest rate scenarios. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.

Agmon, T., Ofer, A. R., and Tamir, A. (1981). Variable rate debt instruments and corporate debt policy. *Journal of Finance* **36**, 113–25.

Aikens, J. S. (1983). Prototypical knowledge for exert systems. *Artificial Intelligence* **20**, 163–210.

Alter, S. L. (1980). *Decision Support Systems: Current Practices and Continuing Challenges.* Addison-Wesley, Reading, Massachusetts.

Andriole, S. (1982). The design of microcomputer-based personal decision-aiding systems. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-12**, 463–9.

Asay, M. R., Bouyoucos, P. J., and Marciano, A. M. (1989). An economic approach to valuation of single premium deferred annuities. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.

Ayers, H. F. and Barry, J. Y. (1979). The equilibrium yield curve for government securities. *Financial Analysts' Journal* **35.3**, 31–9.

Barnea, A., Haugen, R. A., and Senbet, L. W. (1980). A rationale for debt maturity structure and call provisions in the agency theoretic framework. *Journal of Finance* **35**, 1223–34.

Barnea, A., Haugen, R. A., and Senbet, L. W. (1981*a*). Market imperfections, agency problems and capital structure: a review. *Financial Management* 10.3, 7–22.

Barnea, A., Haugen, R. A., and Senbet, L. W. (1981*b*). An equilibrium analysis of debt financing under costly tax arbitrage and agency problems. *Journal of Finance* **36**, 569–81.

Barr, A., and Fergenbaum, E. A., (eds.) (1981). *The Handbook of Artificial Intelligence*, Vol. 1. William Kaufmann, Inc., Los Altos, California.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik* 4, 238–52.

Bierwag, G. O. (1987). *Duration Analysis: Managing Interest Rate Risk.* Ballinger, Cambridge, Massachusetts.

Binbasioglu, M. and Jarke, M. (1986). Domain-specific DSS tools for knowledge-based model building. *Decision Support Systems* **2**, 213–23.

Binbasioglu, M. and M. Jarke (1987). Knowledge-based formulation of linear planning models. In *Expert Systems and Artificial Intelligence in Decision Support Systems* (ed. H. G. Sol *et al*), 113–136. Reidel, Dordrecht.

Birge, J. R. (1985). Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research* **33**, 987–1007.

Birge, J. R., Dempster, M. A. H., Gassmann, H. I., Gunn, E. A., King, A. J., and Wallace, S. W. (1987). A standard input format for multiperiod stochastic linear programs. Mathematical Programming Society *Committee on Algorithms Newsletter* **17**, 1–19.

Black, F., Derman, E., and Toy, W. (1987). A one-factor model of interest rates and its application to treasury bond options. Goldman, Sachs and Co., New York.

Bobrow, D. G. (1984). Qualitative reasoning about physical systems: an introduction. *Artificial Intelligence* **24**, 1–5.

Bodie, Z. and Taggart, R. A. Jr. (1978). Future investment opportunities and the value of the call provision on a bond. *Journal of Finance* **33**, 1187–200.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B. (1982). The evolution from MIS to DSS: extension of data management to model management. In *Decision*

Support Systems (ed. M. J. Ginzberg, W. R. Reitman and E. A. Stohr). North-Holland, Amsterdam.

Boot, J. C. G. and Frankfurter, G. M. (1972). The dynamics of corporate debt management: decision rules and some empirical evidence. *Journal of Financial and Quantitative Analysis* **7**, 1957–65.

Booth, G. G. and Koveos, P. E. (1986). A programming model for bank hedging decisions. *The Journal of Financial Research* **9**, 271–9.

Bouwman, M. J. (1983). Human diagnostic reasoning by computer: an illustration from financial analysis. *Management Science* **29**, 653–72.

Boyce, W. M. and Kalotay, A. J. (1979). Optimum bond calling and refunding. *Interfaces* **9.5**, 36-49.

Bradley, S. P. and Crane, D. B. (1972). A dynamic model for bond portfoli o management. *Management Science* **19**, 139–51.

Bradley, S. P. and Crane, D. B. (1973). Management of commercial bank government security portfolios: an optimization approach under uncertainty. *Journal of Bank Resarch* **4.1**, 18–30.

Bradley, S. P. and Crane, D. B. (1975). *Management of Bank Portfolios.* John Wiley and Sons, New York.

Bradley, S. P. and Crane, D. B. (1980). Managing a bank bond portfolio over time. In *Stochastic Programming* (ed. M. A. H. Dempster), Academic Press, London, 449–71.

Brealey, R., Myers, S., Sick, G. and Whaley, R. (1986). *Principles of Corporate Finance*, First Canadian Edition. McGraw-Hill Ryerson Limited, Toronto.

Brennan, T. J. and Elam, J. J. (1986*a*). Enhanced capabilities for model-based decision support systems. In *Decision Support Systems: Putting Theory into Practice* (ed. R. Sprague, Jr. and H. J. Watson). Prentice-Hall, Englewood Cliffs, New Jersey.

Brennan, T. J. and Elam, J. J. (1986*b*). Understanding and validating results in model-based decision support systems. *Decision Support Systems* **2**, 49–54.

Brick, I. E. and Ravid, S. A. (1985). On the relevance of debt maturity structure. *Journal of Finance* **40**, 1423–37.

Bryant, J. W. (1987). Concepts and techniques of financial modelling. In *Financial Modelling in Corporate Management* (ed. J. W. Bryant), 21–43. John Wiley and Sons Ltd., Chichester.

Coad, P. and Yourdon, E. (1990). *Object-Oriented Analysis.* Prentice-Hall, Englewood Cliffs, New Jersey.

Crane, D. B. (1971). A stochastic programming model for commercial bank bond portfolio management. *Journal of Financial and Quantitative Analysis* **6**, 955–76.

Crane, D. B., Knoop, F. and Pettigrew, W. (1977). An application of management science to bank borrowing strategies. *Interfaces* **8.1.2**, 70–81.

Dantzig, G. B. (1989). Parallel processors for planning under uncertainty. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.

Davis, D. B. (1987). Artificial intelligence goes to work. *HighTechnology* **7.4**, 16–27.

Davis, R. (1984). Amplifying expertise with expert systems. In *The AI Business* (ed. P.H. Winston and K. A. Prendergast), 17–40. MIT Press, Cambridge, Massachusetts.

DeMarco, T. (1978). *Structured Analysis and System Specification.* Yourdon Press, Englewood Cliffs, New Jersey.

Dembo, R. S. (1989). Scenario optimization. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.

Dempster, M. A. H. (1988). On stochastic programming: II. Dynamic problems under risk. *Stochastics* **25**, 15–42.

Dhar, V. and Croker, A. (1988). Knowledge-based decision support in business:

issues and a solution. *IEEE Expert* **3.1**, 53-62.

Dhar, V. and Pople, H. E. (1987). Rule-based versus structure-based models for explaining and generating expert behavior. *Communications of the ACM* **30**, 542-55.

Dolk, D. R. (1990). Structured modelling and discrete event simulation. Unpublished working paper, Naval Postgraduate School, Monterey, California.

Dolk, D. R. and Kottemann, J. E. (1990). Model integration and modelling languages. Unpublished working paper, Naval Postgraduate School, Monterey, California.

Dyer, J. S. and Mulvey, J. M. (1983). Integrating optimization models with information systems for decision support. In *Building Decision Support Systems* (ed. J. L. Bennett), 89-110. Addison-Wesley, Reading, Massachusetts.

Elam, J. J. and Konsynski, B. (1987). Using artificial intelligence techniques to enhance the capabilities of model management systems. *Decision Sciences* **18**, 487-501.

Elton, E. J. and Gruber, M. J. (1971). Dynamic programming applications in finance. *Journal of Finance* **26**, 473-506.

Fabozzi, F. J. and Fabozzi, T. D. (1989). *Bond Markets, Analysis and Strategies.* Prentice-Hall, Englewood Cliffs, New Jersey.

Fabozzi, F. J. and Pollack, I. M. (1983). *The Handbook of Fixed Income Securities.* Dow Jones-Irwin, Homewood, Illinois.

Fedorowicz, J. and Williams, G. B. (1986). Representing modelling knowledge in an intelligent decision support system. *Decision Support Systems* **2**, 3-14.

Fikes, R. and Kehler, T. (1985). The role of frame-based representation in reasoning. *Communications of the ACM* **28**, 904-20.

Filman, R. E. (1988). Reasoning with worlds and truth maintenance in a knowledge-based programming environment. *Communications of the ACM* **31**, 382-401.

Fooladi, I., and Roberts, G. (1990). Bond portfolio immunization: Canadian tests.

Journal of Economics in Business (to appear).

Frank, J. and Schnabel, J. (1983). Timing of borrowing decisions—a decision support system. *Journal of Systems Management* **34.4**, 6–9.

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

Gassmann, H. I. (1987). Multiperiod Stochastic Programming. Unpublished PhD. thesis, Faculty of Commerce and Business Administration, University of British Columbia.

Gassmann, H. I. and Ireland, A. M. (1990). Input/output standards for the MIDAS optimization model. Unpublished working paper WP–90–2, School of Business Administration, Dalhousie University, Halifax, Nova Scotia.

Gassmann, H. I. (1989*a*). Optimal harvest of a forest under uncertainty. *Canadian Journal for Forest Research* **19**, 1267–74.

Gassmann, H. I. (1989*b*). MSLiP: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming* (to appear).

Geoffrion, A. M. (1987). An introduction to structured modelling. *Management Science* **33**, 547–88.

Greenberg, H. J. (1987*a*). A natural language discourse model to explain linear programming models and solutions. *Decision Support Systems* **3**, 333–42.

Greenberg, H. J. (1987*b*). The development of an intelligent mathematical programming system. Paper presented at the Washington Operations Research Science Chapter (WORSC) Meeting.

Greenberg, H. J. (1988). Intelligent user interfaces for mathematical programming. Unpublished paper presented at the Shell Conference on Logistics, Amsterdam, November.

Greenberg, H. J., and Lundgren, J. R. (1989). Extensions of graph inversion to support an artificially intelligent modelling environment. *Annals of Operations Research* **21**, 127–42.

Greenberg, H. J., and Murphy, F. H. (1989). Mathematial infeasibility: criteria for diagnosis. Unpublished working paper, University of Colorado at Denver.

Grinyer, P. H. and Wooller, J. (1975). Computer models for corporate planning. *Long Range Planning* **18.8**, 14-25.

Grove, M. A. (1974). On 'duration' and the optimal maturity structure of the balance sheet. *The Bell Journal of Economics and Management Science* **5**, 696-709.

Hamilton, W. F. and Moses, M. A. (1973). An optimization model for corporate financial planning. *Operations Research* **22**, 677-92.

Hamilton, W. F. and Moses, M. A. (1974). A computer-based corporate planning system. *Management Science* **21**, 148-59.

Hammond, J. S. (1974). Do's and don'ts of computer models for planning. *Harvard Business Review* **52**, 110-23.

Handorf, W. C. (1974). Flexible debt financing. *Financial Management* **3.2**, 17-23.

Henderson, J. C. (1987). Finding synergy between decision support systems and expert systems research. *Decision Sciences* **18**, 333-49.

Heymann, H. G. and Bloom, R. (1988). *Decision Support Systems in Finance and Accounting.* Quorum Books, New York.

Hogue, J. T. and Watson, H. J. (1984). Current practices in the development of decision support systems. *Proceedings of the Fifth International Conference on Information Systems.* Society for Information Management, Chicago.

Howard, C. D. (1986). Applications of Monte Carlo simulation to corporate financial liability management. Unpublished paper presented at the Financial Management Association Conference, New York, October.

Hunter, W. T. (1986). *Canadian Financial Markets.* Broadview Press Ltd., Peterborough, Ontario.

Index Technology Corporation (1989). *Excelerator Facilities and Functions Reference Guide.* Cambridge, Massachusetts.

IntelliCorp, Inc. (1988*a*). *KEE User's Guide.* Mountain View, California.

IntelliCorp, Inc. (1988*b*). *Common Windows Manual.* Mountain View, California.

IntelliCorp, Inc. (1988*c*). *KEEpictures Reference Manual.* Mountain View, California.

IntelliCorp, Inc. (1988*d*). *KEE RuleSystem3 Reference Manual.* Mountain View, California.

Ireland, A. M. (1987). An Expert Decision Support System for Debt Management, Working Paper No. 55. School of Business Administration, Dalhousie University.

Kalotay, A. J. (1980). On the maturity mix of public utility debt. *Public Utilities Fortnightly* **33**, 31–5.

Kalymon, B. A. (1971). Bond refunding with stochastic interest rates. *Management Science* **18**, 563–75.

Keen, P. G. (1976). Interactive computer systems for managers: a modest proposal. *Sloan Management Review* **18.1**, 1–17.

Keen, P. G. (1980). Decision support systems: translating analytical techniques into useful tools. *Sloan Management Review* **21.3**, 33–44.

Keen, P. G. and Scott Morton, M. S. (1978). *Decision Support Systems: An Organizational Perspective.* Addison-Wesley, Reading, Massachusetts.

Klein, R. (1982). Computer-based financial modelling. *Journal of Systems Management* **33.5**, 6–13.

Kosy, D. W. and Wise, B. P. (1986). Overview of ROME: A Reason-Oriented Modeling Environment. In *Artificial Intelligence in Economics and Management* (ed. L.F. Pau), 21–30. Elsevier, Amsterdam.

Lane, M. and Hutchinson, P. (1980). A model for managing a certificate of deposit portfolio under uncertainty. In *Stochastic Programming* (ed. M. A. H. Dempster), 473–95. Academic Press, London.

Lee, C. F. (1985). *Financial Analysis: Theory and Application.* Addison-Wesley, Reading, Massachusetts.

Lee, J. K. (1989). Integration of linear programming with knowledge-based systems by the post-model analysis approach. *Management Science* (to appear).

Lee, J. K., Chu, S. C., Kim, M. Y. and Shim, S. H. (1989). A knowledge-based formulation of linear programming models using UNIK-OPT. In *Proceedings, Second International Workshop on Artificial Intelligence in Economics and Management.* North-Holland, Amsterdam (to appear).

Lee, J. K. and Hurst, E. G. (1988). Multiple-criteria decision making including qualitative factors: the post-model analysis approach. *Decision Sciences* **19**, 334–52.

Lee, J. K. and Kang, B. S. (1988). Intelligent production planning using the post-model analysis approach. In *Applied Expert Systems* (ed. E. Turban and P.R. Watkins), 87–106. Elsevier, Amsterdam.

Lee, J. K. and Kim, H. S. (1989). Intelligent stock portfolio management system. *Expert Systems* **6.2**, 74–87.

Lee, J. K. and Lee, H. G. (1987). Interaction of strategic planning and short-term planning: an intelligent DSS by the post-model analysis approach. *Decision Support Systems* **3**, 141–54.

Markowitz, H. (1959). *Portfolio Selection.* John Wiley and Sons, New York.

McInnes, J. M. and Carleton, W. J. (1982). Theory, models and implementation in financial management. *Management Science* **28**, 957–78.

Meador, C.L., Keen, P. G. W., and Guyote, M. J., (1984). Personal computers and distributed decision support. *Computerworld* **18.19**, ID7–ID16.

Meador, C. L. and Ness, D. N. (1974). Decision support systems: an application to corporate planning. *Sloan Management Review* **15.2**, 51–68.

Miller, L. W. and Katz, N. (1986). A model management system to support policy analysis. *Decision Support Systems* **2**, 55–63.

Minsky, M. (1975). A framework for representing knowledge. In *The Psychology of*

Computer Vision (ed. P. Winston), 211–77. McGraw Hill, New York.

Modgliani, F. and Miller, M. (1958). The cost of capital, corporation finance and the theory of investment. *American Economic Review* **48:3**, 261–97.

Modgliani, F. and Miller, M. (1963). Corporate income taxes and the cost of capital: a correction. *American Economic Review* **53.3**, 433–43.

Morris, J. R. (1976*a*). On corporate debt maturity strategies. *Journal of Finance* **31**, 29–37.

Morris, J. R. (1976b). A model for corporate debt maturity decisions. *Journal of Financial and Quantitive Analysis* **11**, 339–57.

Mulvey, J. M. and Vladimirov, H. (1988). Solving multistage stochastic networks: an application of scenario aggregation. Unpublished technical report SOR–88–1, Department of Civil Engineering and Operations Research, School of Engineering and Applied Science, Princeton University, Princeton, New Jersey.

Mulvey, J. M. and Vladimirov, H. (1989*a*). Stochastic network optimization models for investment planning. In: *Annals of Operations Research: Network Optimization and Applications*, v.20 (ed. B. Shetty). J.C. Baltzer, 187–217.

Mulvey, J. M. and Vladimirov, H. (1989*b*). Solution techniques for multi-scenario generalized network programs. Unpublished technical report SOR–89–9, Department of Civil Engineering and Operations Research, School of Engineering and Applied Science, Princeton University, Princeton, New Jersey.

Murphy, F. H. and Stohr, E. A. (1986). An intelligent system for formulating linear programs. *Decision Support Systems* **2**, 39–47.

Naylor, T. H. (1976). A conceptual framework for corporate modeling and the results of a survey of current practices. *Operational Research Quarterly* **27**, 671–82.

Naylor, T. H. (1983). Strategic planning models. *Managerial Planning* **30**, 3–11.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, New Jersey.

Nova Scotia Power Corporation (1985). Internal job descriptions for Treasurer and

Treasury staff. Halifax, Nova Scotia.

Nova Scotia Power Corporation (1990). *Annual Report.* Halifax, Nova Scotia.

Page-Jones, M. (1988). *The Practical Guide to Structured Systems Design*, Second Edition. Yourdon Press, Englewood Cliffs, New Jersey.

Power, P. D. (1975). Computers and financial planning. *Long Range Planning* **8**, 53–9.

Pressman, R. S. (1987). *Software Engineering: A Practitioner's Approach*, Second Edition. McGraw-Hill, New York.

Prisco, S. (1989). Valuing derivative securities using Monte Carlo simulation. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.

Quillian, M. R. (1968). Semantic memory. In *Semantic Information Processing* (ed. M. Minsky), 227–70. MIT Press, Cambridge, Massachusetts.

Raiffa, H. (1968). *Decision Analysis: Introductory Lectures on Choices under Uncertainty.* Addison-Wesley, Reading, Massachusetts.

Scott-Morton, M. S. (1971). *Management Decision Systems: Computer-Based Support for Decision Making.* Division of Research, Harvard University, Cambridge, Massachusetts.

Shapiro, Jeremy F. (1988). Stochastic programming models for dedicated portfolio management. In: *Mathematical Models for Decision Support* (ed. G. Mitra), 587–611. NATO ASI Series F **48**. Springer, Berlin.

Shim, J. and McGlade R. (1984). The use of corporate planning models: past, present and future. *Journal of the Operational Research Society* **35**, 885–94.

Simon, H. (1977). *The New Science of Management Decisions*, Revised Edition. Prentice-Hall, Englewood Cliffs, New Jersey.

Sivasankaran, T. and Jarke, M. (1985). Logic-based formula management strategies in an actuarial consulting system. *Decision Support Systems* **1**, 251–62.

Sprague, R. H. Jr. (1980). A framework for the development of decision support systems. *MIS Quarterly* **4.4**, 1–26.

Sprague, R.H. Jr. and Carlson, E. D. (1982). *Building Effective Decision Support Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Stansfield, J. L. and Greenfeld, N. R. (1987). PlanPower: a comprehensive financial planner. *IEEE Expert* **2.3**, 51–60.

Steele, G. L. (1984). *Common LISP: The Language.* Digital Press.

Stefik, M. J. and Bobrow, D. G. (1986). Object-oriented programming: themes and variations. *The AI Magazine* **6.4**, 40–62.

Stefik, M. J., Bobrow, D. G. and Kahn, K. M. (1986). Integrating access-oriented programming into a multiparadigm environment. *IEEE Software* **3.1**, 10–18.

Tello, E. R. (1989). *Object-oriented Programming for Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts.

Texas Instruments Incorporated (1987). *Common Windows Manual.* Austin, Texas.

Thatcher, J. S. (1985). The choice of call provision terms: evidence of the existence of agency costs of debt. *Journal of Finance* **40**, 549–61.

Turban, E. (1988). *Decision Support and Expert Systems.* Macmillan, New York.

Turban, E. and Watkins, P. R. (1986). Integrating expert systems and decision support systems. *MIS Quarterly* **10.2**, 121–36.

Van Slyke, R. and Wets, R. J.-B. (1969). L-shaped linear programs with application to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics* **17**, 638–63.

Vasicek, O. A. and Fong, H. G. (1982). Term structure modeling using exponential splines. *The Journal of Finance* **37**, 339–48.

Viscione, J. A. and Roberts, G. S. (1987). *Contemporary Financial Management.* Merrill, Columbus, Ohio.

Waterman, D. A. (1986). *A Guide to Expert Systems.* Addison-Wesley, Reading, Massachusetts.

White, W. L. (1974). Debt management and the form of business financing. *Journal* *of Finance* **29**, 565–77.

Whitten, J. L., Bentley, L. D., and Barlow, V. M. (1989). *Systems Analysis and Design Methods*, Second Edition. Richard D. Irwin, Homewood, Illinois.

Winkelbauer, L. (1988). A structure for the incremental construction of coupled systems. In *Coupling Symbolic and Numeric Computing in Expert Systems* (ed. J. S. Kowalik and C. T. Kitzmiller), 145–56. North-Holland, Amsterdam.

Wise, B. P. and Kosy, D. W. (1986). Model-based evaluation of long-range resource allocation plans. In *Artificial Intelligence in Economics and Management* (ed. L. F. Pau), 93–102. Elsevier, Amsterdam.

Zenios, S. A. (1989). Parallel Monte Carlo simulation of mortgage backed securities. Unpublished paper presented at the Financial Optimization Conference, Wharton School, University of Pennsylvania, November.