INFORMAL, SEMI-FORMAL, AND FORMAL APPROACHES TO THE SPECIFICATION OF SOFTWARE REQUIREMENTS

By

Marie Hélène LinLee Wong Cheng In

B. Sc. (Computer Science) Angelo State University 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

[illegible]

[illegible]

[illegible]

THE UNIVERSITY OF BRITISH COLUMBIA

August 1994

Abstract

The derivation of a specification document that is understandable, precise and unambiguous is indispensable to successful software development. This work investigates the advantages and disadvantages of four different specification approaches which vary in their degree of formality. The thesis outlines qualities of a good specification. An example requirements specification of a case study system is produced using each technique. The four specification approaches are *ad hoc* natural language approach (informal), threads-based technique (structured informal), Computer-Aided Software Engineering (CASE) methodology Structured Analysis (semi-formal), and the mathematical notation Z (formal). The specification techniques are compared based on their likelihood to produce a specification document which is useful to customers, design engineers, test engineers, and maintenance personnel.

The main conclusion is that using techniques that provide both a high degree of guidance and process description for deriving the specification is critical to achieve high quality specifications. Hence, a good specification technique must inherently have guidelines that facilitate the specification of requirements in an understandable, precise and unambiguous manner.

Table of Contents

List of Tables

List of Figures

Acknowledgement

Thank you Jeff for being a great supervisor and for your continuous support and encouragement during the past year and a half. Jeff, I have enjoyed working with and for you. Experience at Hughes Aircraft Systems Division was invaluable in many aspects of this research. Many thanks go to Dr Karl Toth who made it possible for me to have access to the work that Hughes is doing and for his comments on my thesis.

My thanks to Jonathan Masters for much of his valuable time to tell me about specification in industry. Thank you Jon for the quick turnaround time and for your useful comments and insights. I would also like to acknowledge Dave Clark, Wayne Powell, and Don Chandler for meeting with me to discuss industrial concerns on specification. They encouraged me and believed in the practical value of my thesis.

Xiaomei Han, Tony Lau, Vishwa Ranjan and Scott Hazelhurst are to be thanked for their support and constant encouragement. Without them, I would not have survived my first year at UBC. Thank you Mike McAllister, Scott Hazelhurst, Alistair Veitch, Carl Alphonce, Zheng Zhu, Peter Smith and Russell Burnett for reading drafts of this thesis and for helping me write a better thesis.

I am indebted to Russell Burnett for listening to my daily rantings and ravings about my progress in my thesis. Thank you Russell for believing in me and for encouraging me through graduate school. Thank you for putting up with me during the last few months of the writing up. Thank you for putting a little bit of sunshine in my heart when things were rough.

Special thanks go to Scott Hazelhurst who has always been there to listen to me, to give me advice, and encourage me with my school work and throughout the process of writing my thesis. Thank you for reading more than your share of drafts of the thesis and for always being ready to help me write better sentences. Thank you for being a true friend.

Warm and fuzzy thanks to Catherine Leung for sharing many cheesecakes and giggles with me and for introducing me to sushi and sashimi. Warm and fuzzy thanks also to Xiaomei Han for being a good friend and for our long talks, sharing some of her meals with me and for watching movies with me. Thank you both for the many good laughs and for caring.

A word of appreciation to many people in the department who have brightened my days at UBC: Marcelo Walter, Peter Smith, Jim Boritz, Eric Borm, Mike Donat, Thomas Ligocki, Mohammad Darwish, Ann Lo, Xiaomei Tian, Jinhai Yang, Parag Jain and members of the ISD laboratory. Thank you all for making me feel that life is good.

Special thanks to my office mates, Xiaomei Han, Jim Boritz, and Momo Darwish who gave me privileged use of the computer terminal during the writing up of my thesis. Thank you for the friendly and homey atmosphere in the office and the occasional chit chats.

My aunt and uncle Claire and Gerard, and Kevin and Karine have been very encouraging, helpful and supportive during my stay in Vancouver.

I gratefully acknowledge the research grants from the Natural Sciences and Engineering Council of Canada and the British Columbia Advanced Systems Institute which supported this research.

My greatest debt is that due to my parents. Thank you Mamie and Pa for trusting me and encouraging me to dream big dreams. Thank you Polo, Laval, and Georges for believing that I can do anything that I set my heart to. Thanks to all my friends at home who have never faltered in their belief of me.

To Mamie and Pa

Chapter 1

Introduction

1.1 Motivation

The need to state requirements correctly, precisely and completely early in the software development cycle is essential to successful software development. Boehm [1] reports that there are quantitative benefits gained from resolving requirement problems in the early phases of the life cycle. The later that defects are found in the requirements specification, the more expensive the cost of the software development. Since the initial task of deriving the requirements drives the later stages of the software development process, it is crucial to establish the requirements in an unambiguous and precise manner.

1.2 Overall Method

This research investigates the advantages and disadvantages of four different types of specification techniques and their suitability to various types of systems. The goal of this research is to provide some demonstration of analysis and specification applied to a common problem, and ultimately, to assist those unfamiliar with the various specification techniques to choose an appropriate one for specifying their systems. Combining the various techniques to ensure high levels of success in understanding and specifying system requirements is a second area of interest.

More specifically, I have taken a non-trivial system, namely the telephone registration system (Telereg[1]) at the University of British Columbia (UBC) as a case study. I have chosen four different specification techniques that vary in their degree of formality and I have produced an example requirements specification of the Telereg system using each technique. The techniques are compared with respect to a number of qualities and factors including their readability, the potential for ambiguity, the amount of technical skill required, the use and availability of any associated tools, and the difficulty of analysis.

The four techniques chosen are:

- *ad hoc* natural language approach
- threads-based technique
- CASE methodology Structured Analysis
- formal specification using Z

These four techniques were selected because they are representative of the range of specification techniques. Both the *ad hoc* natural language and the threads-based approaches are informal types of specification techniques because they use natural language as their notational tool. Structured Analysis is a semi-formal type of specification technique because it combines natural language with a graphical notation with defined semantics. The specification language Z is a formal technique because it is founded on mathematical principles with a well-defined syntax and semantics. The natural language approach requires the least amount of technical skill but it also has the highest potential for ambiguity and difficulty of analysis. At the other end of the spectrum, the formal specification technique requires the largest amount of technical skill, but it is the most amenable to analysis and has the least potential for ambiguity.

[1] *TELEREG*$_{TM}$ is a registered trademark of the University of British Columbia.

The main conclusion is that using techniques that provide both a high degree of guidance and process description for deriving the specification is critical to achieve high quality specifications. Hence, a good specification technique must inherently have guidelines that facilitate the specification of requirements in an understandable, precise and unambiguous manner and that facilitate the systematic derivation of the specification. Then, the technique can contribute to producing a requirements specification that is understandable and beneficial to its users.

1.3 The SIS-Telereg System

The **Student Information System** (SIS) at UBC is designed to establish a single database for student information. It allows decentralized update and inquiry access with appropriate access controls, and provides convenient interfaces to other centralized administrative systems. The Student Information System also provides data to support comprehensive reporting and analysis as well as the flexibility to implement new policies and support changing needs of the University, faculties and departments. **Telereg** is UBC's telephone registration system which was introduced in 1988. Telereg is the student's interface to the SIS. It allows students to register for their courses from any touch-toned telephone. The students use the buttons on the telephone keypad to enter requests and the computer's voice guides them through their registration. The students may request to add a course, drop a course, change a course section, confirm a section switch, cancel their entire registration, list all their courses, and other related requests.

1.4 *Ad hoc* Natural Language Specification Approach

The *ad hoc* natural language specification approach is an informal type of specification that is predominantly used in industry today. This approach is often described in software engineering textbooks. It organises requirements into certain predefined sections; each section is written in natural language using a free style format. Certain rules describe the expected content or scope of each section but there are few guidelines on how the requirements are written, except for standards such as the DOD-STD-2167A and ANSI/IEEE Standard 830. The most commonly used sections include an introduction, system context, system evolution, functional requirements, non-functional requirements, validation criteria, and glossary [29] [26].

The introduction section describes the functionality of the system, the rationale for the system, and its contribution to the organisation commissioning the system. The system context section specifies the environment in which the system occurs; it includes the physical, hardware, peripheral, and software contexts. The system evolution section describes anticipated changes to the hardware, software, and peripheral components. The functional requirements describe what the system will do, that is, its desired behaviour. The constraints imposed on the system appear in the non-functional requirements section of the specification document. These constraints may include the programming language to use, reliability requirements, security, concurrency, and response time. The validation criteria section describes information related to performance bounds, classes of tests, and expected software responses. The glossary section defines the technical terms used in the document.

Overall, writing requirements in natural language is *ad hoc* and open ended. This approach is subject to ambiguity and imprecision because these properties are inherent in the "notational tool" (natural language). Also, in a large software project, this approach can result in various specification styles throughout the specification document since there are very few guidelines to assist the various writers in their efforts. However, this approach requires the least amount of technical skill. Any type of analysis such as test case generation is a labour intensive task. Natural language understanding tools may be used to parse an *ad hoc* natural language specification for analysis but the results might not have a high level of success because of the lack of structure in the specification document.

1.5 Structured Analysis

Structured Analysis is a semi-formal type of specification technique which combines the use of natural language and graphical symbols with semantics. Structured Analysis is a methodology that describes a system as a function from inputs to outputs. The methodology has a well-defined syntax and a loosely defined semantics. Structured Analysis offers a graphical data flow model of a system and uses the divide and conquer principle to depict a hierarchical high level view of the system. Consequently, Structured Analysis consists of various levels of abstraction with each level placing emphasis on transformation of data. Each level of abstraction is called a data flow diagram and it depicts the functional decomposition of the system from the point of view of data rather than from the viewpoint of one user or a group of users. A data flow diagram is essentially a picture showing the flow of information and how that information is processed by transformation centres called processes. Structured Analysis may be used to specify control behaviour as well as information processing.

The top-most level of abstraction is a special case of a data flow diagram called a context diagram. The context diagram describes the system in relation to its environment. It shows the net inputs and outputs of the system. Each of the intermediate levels is a detailed decomposition of a process from a higher level. The bottom-most level consists of a series of natural language specifications of processes that are not further decomposed. Descriptions of terms used in the data flow diagrams appear in a data dictionary.

In Structured Analysis, graphical layouts enhance human readability and comprehension thereby improving communication. When compared to a natural language technique, Structured Analysis has less potential for ambiguity but requires a greater amount of technical skill. This methodology has tool support which can undertake the task of automatically checking the syntactic consistency and completeness of the model. Tools that assist in system design are available and widespread. Structured Analysis offers a top down analysis of the system. Therefore, a model system can be read at different levels of detail. The lack of precise semantics gives the flexibility of specifying the system without having to worry about details in the notation.

Structured Analysis not only causes the analyst to specify the requirements but to partition what is being specified as well, thus increasing the risk of encountering design details. Partitioning introduces design structure in the model. When design decisions are made during later stages, the design structure must be changed. So, a Structured Analysis model is not easily maintainable. Moreover, the lack of precise semantics in the methodology still allows for ambiguities in the specification model to remain undetected.

1.6 Threads-Based Technique

The threads-based technique is a structured informal specification technique; it is a refinement of the natural language approach. It is based upon the concept of a *thread* which is defined as "a path through a system that connects an external event or stimulus to an output event or response" [12]. A thread-based specification consists of a series of statements, each relating an external stimulus to an external response. The threads-based specification technique applies only to the functional requirements section of a system requirements document. It is a methodology that partitions the functional requirements of a system into basic specification units called *threads, capabilities,* or *conditions* and describes components of the system state in a *data dictionary.*

The threads-based approach provides a functional view of the system where a stimulus results in a response or an update in the system state. A set of documented conventions govern the functional specification which, written in natural language, uses no mathematical or graphical notations. More importantly, the threads-based approach makes the stimuli to and responses from the system explicit and mandates the use of "shall" statements for any requirements that are testable. The requirements are written from an external point of view. The specification units and the data dictionary are the notational tools of this technique.

The threads-based specification technique offers the advantage of writing requirements in a less *ad hoc* manner than the natural language approach. The threads-based technique is potentially less ambiguous than the *ad hoc* natural language approach because of the added structure and writing conventions imposed by the technique. The rules of convention and writing style also provide a systematic and thus standard way of expressing requirements. The amount of technical skill involved in this technique is greater than required in the *ad hoc* approach. The added structure facilitates analysis such as the generation of test cases. While there might not be any tool support to check the consistency of the specification effort, there exists tool support to parse threads-based text and extract information from it using intelligent analysis.

1.7 Formal Specification Language Z

Specification language Z (pronounced "zed") is considered a formal method based on its definition as "a general description of the use of mathematical notations such as logic and set theory to describe system specifications and software designs together with techniques of validation and verification based on mathematics" [17]. Z is a machine readable mathematical notation based on set theory and first order logic. Z is used to specify a mathematical model of the system whereby the problem domain consists of data objects and the relationships between the various data objects in the system are described by the requirements. The data objects are made of elements called basic types. The building blocks for the basic types are sets and functions. Tools are available to provide automatic checking for type consistency across the specification.

A specification written in Z is a mixture of formal mathematical statements and informal text. The formal mathematical statements give a precise description of the system, in terms of operations performed on the data objects, by specifying the system state before processing an event and after processing the event. The informal text describes in natural language the meaning of the mathematical statements to make the specification more readable. The formal statements are grouped into packaging mechanisms called *schemas.* Each schema consists of a declaration part that declares the variables used in the schema's operation and a body part that describes the operations. The schemas are abstraction modules and they are combined to produce a specification of the system.

The Z specification of the SIS-Telereg system was derived from its threads-based specification because the latter lends itself rather naturally to be formalized in Z. The structure of a threads-based specification is similar to that of a Z specification.

Formal specifications require us to write down user requirements very precisely, thereby decreasing the potential for ambiguity. However, this technique requires a high level of technical skill. The type checking support offered by Z ensures the consistency of a Z specification. Manual checking is required to maintain consistency between the mathematical statements and the informal text. Since a specification in Z is machine-readable, it can be used as input to some form of analysis, be it for formal validation, formal verification, or for the generation of test cases during system testing. Some analysis, simplified by formal techniques, helps us to achieve a high degree of confidence that a system conforms to its specifications. Writing a formal specification enables us to find loopholes and inconsistencies in the requirements. However, it is also possible to write mathematical statements in Z that are correctly type checked but have no meaning.

1.8 Thesis Outline

Chapter 2 reviews what a requirements specification document is and examines its uses and its characteristics. It is important that we understand the purposes and uses of a requirements specification so that we can define its characteristics, and thus, be able to appreciate what we want to achieve as a result of specifying a system.

Chapter 3 presents the motivation for using the SIS-Telereg system as the case study and describes the multi-step approach used to conduct the research. Some of the observations made during the course of this work are stated. Chapter 3 also justifies the motivation for the organisation of the following four chapters.

Chapters 4, 5, 6, and 7 respectively describe the *ad hoc* natural language approach, the semi-formal CASE methodology Structured Analysis, the structured informal threads-based technique, and the formal specification notation Z. Each of these chapters present the intrinsic properties, conventions, rules of thumb, and any available tool support for each technique. In addition, Chapters 4 to 7 demonstrate how parts of the SIS-Telereg system can be specified using each technique. Each chapter concludes with an evaluation of the approach.

Finally, Chapters 8 and 9 present the results and the overall conclusions for this work.

Chapter 2

Requirements Specification

2.1 Introduction

The necessity to invest in requirements analysis cannot be overstated. While efforts in the past have concentrated on improving structured programming, now they are spent on the front end of the software life cycle [12]. It is not uncommon to read about system products that are not delivered on time or completed within budget. Chandler [3] mentions cases with cost and time scale overruns of 200% to 300%. Often, the cause has been that errors are found in the later stages of the software development life cycle. Thus, it is highly important that requirements be analysed and understood properly so that ambiguities and loopholes can be found at the start of the project. The sooner errors are found, the less costly they are. Ideally, companies should spend about 20 percent of their total budget on requirements analysis and specification [5] [21]. Given that this first task in the software life cycle is crucial to the later stages, it is therefore important that the main deliverable product of requirements analysis, namely the Requirements Specification, be established properly so that it is valuable and useful to the later stages.

2.2 Requirements Analysis and Specification

At the beginning of most software projects, the customer provides the contractor with a statement of requirements. Typically, the customer statement of requirements is expressed in natural language using technical terms of the customer application domain.

The contractor will derive a Requirements Specification from the customer statement of requirements such that the former is subordinate to the latter but it describes the requirements in more details [5]. The process of identifying and understanding the customer requirements is called *requirements analysis*. The analysis phase also requires that the contractor determine the feasibility of the proposed system. The Requirements Specification is the end product of the requirements analysis phase in the software development life cycle.

Rumbaugh et al. [27] classify systems into different categories – batch transformation, continuous transformation, interactive interface, dynamic simulation, real-time system, and transaction manager. Since not all software systems are the same kind, a specification technique that is suitable for one kind of system may not be appropriate for another. For example, a control dominated system such as a traffic light system may be specified using Statecharts (a graphical extended state transition notation) [9]. On the other hand, Statecharts may not be suitable for an information dominated system such as the SIS-Telereg system, the case study of this thesis. Also, systems may be of different sizes and a technique that is suitable for a small system may not be suitable for a large system. There have been instances where commercial CASE tools have lacked the capacity to store a complete model.

2.3 What is a Requirements Specification?

The Requirements Specification describes both the functional and non-functional requirements of the system. The functional requirements describe the services that the system is expected to provide by specifying how inputs to the system are transformed into outputs. Functional requirements detail the inputs and their validation as well as system state changes and system responses to both normal and error conditions. Non-functional requirements describe the constraints under which the system must operate and any design restrictions imposed on the system. More specifically, non-functional requirements may include details on specific response times, memory requirements, hardware limitations, programming language to use, and any mandatory standards.

The Requirements Specification describes *what* a system must do as opposed to *how* these tasks must be accomplished. The document must contain a clear, correct, and unambiguous statement of the intended function of the software and must not describe more than what the system has to do. The Requirements Specification only expresses the external behaviour of the system and should not contain any software design details except for the required constraints and implementation considerations imposed by the customer. Requirements related to safety, quality assurance, and test methodology are specified. Additionally, the document must contain external interface requirements, system evolution, and exceptional events requirements. Requirements that describe goals, objectives, or internal processing must be avoided. For example, it is not useful to state "the graphical user interface shall be user-friendly". The customer and contractor may have different notions about what makes a system user-friendly. Such a requirement is also hard to test. Rather, it is more useful, for instance, to provide some window snapshots of what the graphical user interface must look like when certain functions are requested.

To produce a valuable Requirements Specification, it is important that we first understand how it is generated and what its uses are. Then, once we understand its purposes, we can better appreciate how the specification must be written and what the specification's characteristics must be.

Section 2.4 states the uses of a Requirements Specification. The characteristics of a useful Requirements Specification are discussed in Section 2.5. Section 2.6 describes some types of analysis which may be done on a Requirements Specification.

2.4 Uses of a Requirements Specification

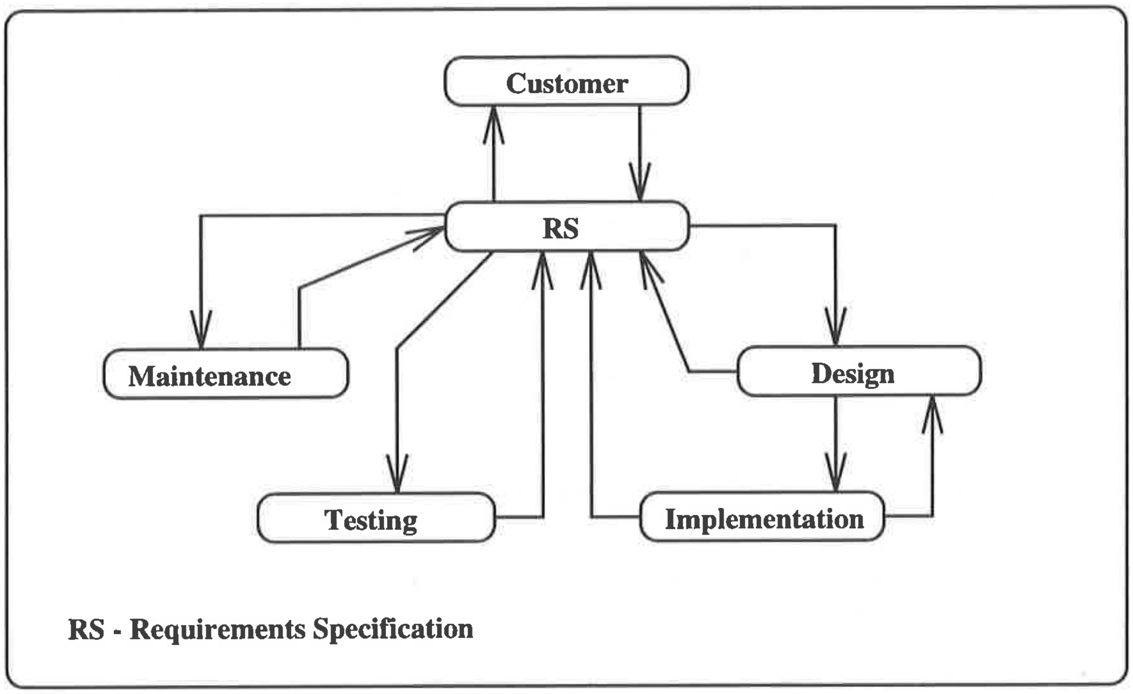


Figure 2.1: Requirements Specification as a base document

The Requirements Specification is often viewed as a contractual agreement between the customer and the contractor in that it specifies what the contractor is expected to deliver at the end of the project.

In addition, the Requirements Specification is the base document which is used for all subsequent developmental activities (see Figure 2.1). The Requirements Specification is derived from the customer statement of requirements, and as the requirements are analysed and understood, if the product requested by the customer is not economically or technically feasible or if some requirements are conflicting, the customer statement of requirements is changed. The Requirements Specification also drives design and testing. If design and test engineers find that their tasks are made difficult because of the way some requirements are specified, the document has to be changed. In addition, the

Requirements Specification is used in maintenance because it documents the functionality of the system. Since the Requirements Specification is essential to the successive stages in the software life cycle, it is highly important that the requirements are understood and well documented so that development efforts are reduced for the later stages. The Requirements Specification provides a baseline for validation and verification [32].

Moreover, the Requirements Specification is used to estimate costs and schedules. Since it gives a clear description of what the system must do, it provides a basis for metrics so that progress made in the system development can be measured [5] [21].

2.5 Qualities of a Useful Requirements Specification

A useful Requirements Specification has the desirable qualities of being:

- understandable
- unambiguous
- complete
- partitioned
- verifiable
- consistent
- modifiable
- traceable
- testable
- maintainable

- decoupled from design
- concise

The production of the Requirements Specification is a joint effort between the customer and the contractor, where the customer typically provides the domain knowledge and the contractor provides the software development expertise. Hence, the specification must be **understandable** to the customer because it is the contractual agreement between both parties. The contractor has to validate all the requirements with the customer during review sessions, and therefore, the object of the review, namely the Requirements Specification, must be accessible to the customer. In addition, a specification that is understandable to the customer is valuable because it can give the customer confidence that the contractor understands what the customer would like built. The specification must also be understandable to an independent testing organisation so that independent testing can be performed on the system.

A good specification must be **unambiguous** so that its requirements can be understood as they are intended. Every requirement must have only one possible interpretation. Requirements that are specified in natural language are potentially ambiguous because many interpretations may be associated with them.

A specification must be **complete**; all the requirements necessary to build the product must be included. Any checks made on the inputs of the system and responses to both normal and error conditions must be specified. The specification must conform to any mandatory standards imposed on it. All graphs, figures, and diagrams must be labelled and referenced.

The specification must be **partitioned** in a logical manner to assist the reader or user of the document. Partitioning is especially important for non-trivial projects so that the specification document is readable and accessible to its users. The specification may be partitioned at the document level and at the requirements level [17]. At the document level, the specification is organised into various sections whereby each section details a particular aspect of the Requirements Specification. For instance, there are sections for functional requirements, non-functional requirements, data definitions, and hardware requirements. All the requirements related to a particular concern can be found in a single location. At the requirements level, the specification may be organised in a top-down manner whereby the system may be understood at different levels of detail.

Each and every requirement in the specification must be **verifiable** in that there is "some finite cost-effective process with which a person or machine can check that the software product meets the requirement" [32]. Hence, requirements must be written in such a way that they can be verified. For instance, a requirement such as "The system shall recover as soon as possible" is not verifiable because the phrase "as soon as possible" is not quantified. Phrases such as "usually", "most of the times" and "fast" are vague and not verifiable and must therefore be avoided. If a requirement is verifiable at a later stage in the development process, the point at which it can be verified must be indicated.

A specification must be **consistent** in that the statements in the document do not conflict internally and externally. External conflicts refer to inconsistencies with other documents which are associated with the specification. IEEE Std 830-1984 [32] mentions three types of internal conflicts in a specification: different terms may be associated with the same object; conflicts may arise by attaching different characteristics to the same entity at two different instances where the entity is described; and there may be logical conflicts in the requirements.

The specification must be **modifiable** so that any changes made to the requirements can be made "easily, completely, and consistently" [32]. This implies that the same requirement must not appear in more than one place so that failure to make changes at all the places where it appears does not result into an inconsistent document. Modifications are facilitated if the document contains a useful index, table of contents and cross references.

A specification must have both forward and backward **traceability**. Forward traceability (also called "allocation") refers to associating each requirement in the specification with a unique number and referencing each requirement in other documents derived from the specification. On the other hand, backward traceability refers to having each requirement explicitly referencing its source in documents from previous stages of development. The traceability properties of a specification are important because they facilitate changes made across various developmental stages.

The specification must be a **testable** document. The requirements are to be written such that a test engineer can derive test cases from the requirements independent of the notation used to express the requirements. Hence, the requirements must be specified from a black box view of the system.

The specification must be **maintainable** such that if and when requirements change as a result of a misunderstanding or system evolution, the requirements may easily be modified to reflect the change. The specification must also document how changes in some components of the system might affect other components. This is especially useful since maintenance personnel are frequently not associated with the development of the system.

The specification must be **decoupled from design**, otherwise any decisions made during the design phase will affect the specification, and maintenance problems are worsened.

The specification must also be **concise**. The description of what the system has to do must be complete but not too wordy.

2.6 Types of Analysis on a Requirements Specification

Various types of analysis must be done on the Requirements Specification. These types of analysis may be done manually or automatically using computer-based tools.

If not all the requirements are specified, then as a result of an incomplete specification, the product is also incomplete and is not the product that the customer wanted built. The specification may also be a source of potential error for the later stages of development. Further, the product cannot be tested. For all these reasons, the customer may not accept the product. It is therefore important to determine that all the requirements for the system have been specified. Forward and backward traceability can help ensure that the specification is complete and show how the requirements are linked to previous and future documents.

Analysis to ensure the internal consistency of the document is also necessary. Some automatic checking of the consistency of the data entities in the specification is possible. Manual checking is required to verify that requirements throughout the specification are consistent with each other, that a defined piece of data is really used in the specification, and that all inputs have outputs. Design engineers can informally check the consistency of the specification when they plan the system architecture. Test engineers must independently analyse the specification to determine whether it is a testable document and to generate test cases. Testing is necessary to determine that the final product meets its specifications.

Chapter 3

Research Process and Initial Observations

3.1 Outline

This chapter describes how the research for this thesis was carried out and records some of the observations that were made during the course of doing the work. Section 3.2 justifies the choice of the SIS-Telereg system as the case study for this work. Section 3.3 describes the multi-step approach used to conduct this research and some of the difficulties encountered. Section 3.4 reports some preliminary observations, namely the importance of establishing a process for requirements specification. Section 3.5 states the motivation for the document organisation in Chapters 4 to 7. Finally, Section 3.6 discusses some general observations made during this work.

3.2 Choice of Case Study: SIS-Telereg System

While the SIS-Telereg system is a non-trivial system, it is an appropriate case study because it is simple enough to explain and compare four different specification approaches. It is also a real system that involves a variety of input sources, a variety of different kinds of real-world entities, and a variety of different kinds of functions. At the same time, the problem domain was familiar and both the customers (UBC administration) and developers of the system (UBC computing production and support groups) were close at hand.

3.3 Research Method

This section presents the muti-step approach used to conduct this research and some of the difficulties encountered.

3.3.1 Requirements Analysis

Since the actual requirements specification for the SIS-Telereg system is proprietary to the University of British Columbia and, hence, not available to the general public, the first task of this research consisted of reverse-engineering the system to understand its requirements. Information about the system was gained from the "UBC Registration Guide" and interviews with an administrator at the Office of Registrar at UBC and system developers from both the production and the support groups. These interviews were very important because the users (students) see a very simplified view of the operation of the system. Because the telephone is the main external interface to the SIS-Telereg system, inquiries were made to the local telephone company. Had the system specification for the SIS-Telereg system been non-existent, then this analysis exercise with the customers and developers would have been used to derive the system specification. For the purposes of this research, we assume that the system specification for the SIS-Telereg system does not exist. With this assumption, it is proposed that the specification be derived for maintenance support.

3.3.2 Choice of Specification Techniques

In parallel with the requirements analysis task of this research, the range of available specification techniques were examined to select four techniques representative of the range. As discussed in Chapter 1, the four techniques chosen were *ad hoc* natural language approach, Structured Analysis, threads-based technique, and formal language Z.

The research consisted of learning these techniques and deriving an instance of the specification of the SIS-Telereg system using each technique. The four specification documents can be found in appendices A to D of this thesis for closer inspection.

3.3.3 *Ad hoc* Informal Specification Approach

The specifications in free style natural language and Structured Analysis were attempted in parallel. The overhead for learning the natural language approach was mainly in understanding how to allocate the system requirements to the appropriate classes of requirements, in particular, functional and non-functional. For instance, requirements describing reliability issues belong to the non-functional requirements. This specification was achieved by using a sample specification of another system as example and by acquiring information about the SIS-Telereg system. Several rewrites of the specification were required as the system requirements became better understood. Organising the functional requirements presented the most problems because they form the bulk of the requirements, but this task was facilitated after the functional requirements were specified in Structured Analysis. The *ad hoc* natural language specification gives a comprehensive view of the system: its purpose, context, functional requirements, non-functional requirements, anticipated evolution, validation criteria, and system behaviour during exceptional events.

3.3.4 Semi-formal CASE Structured Analysis Methodology

My explanation and critique of Structured Analysis are based on the books by DeMarco [10] and Hatley/Pirbhai [15]. I have also gained experience in Structured Analysis by instructing undergraduate students in the methodology and helping them develop their own models for their course case study. Learning this methodology had a higher overhead than the natural language approach because it has a more sophisticated notation with defined semantics and it is supported by commercial tools. The principles of pure Structured Analysis are intuitive; on the other hand, its real time extensions can be harder to grasp and to apply effectively. Using Structured Analysis helped understand and organise the functional requirements of the system so that once they were partitioned, the task of specifying them in natural language for the *ad hoc* requirements document was greatly facilitated and systematic. The Structured Analysis specification addressed only the functional requirements of the SIS-Telereg system because the technique is not intended to capture non-functional or contextual (physical, peripheral, and hardware) requirements.

3.3.5 Structured Informal Threads-Based Technique

The threads-based natural language specification technique is not a technique that is taught in software engineering textbooks. I became aware of this technique during my association with Hughes Aircraft of Canada Ltd, Systems Division (HCSD) in a research project investigating the possible applications of formal methods to large software systems. In my involvement with HCSD, I was exposed to their software requirement specifications and it became very clear that their natural language software specification technique was highly beneficial in that it added structure to the specification. The value of the threads-based technique was only truly appreciated when the functional requirements of the SIS-Telereg were specified using the technique. Although I had thought that the *ad hoc* specification to be quite acceptable and complete, several flaws were uncovered in it. Since the system is of intermediate complexity, the logical organisation of the requirements in the *ad hoc* specification mapped nicely into that of the threads-based one. In larger systems, more work would be needed to allocate the requirements into the specification units of the technique. The biggest overhead with this technique was studying examples of threads-based specification and determining the properties that were absolutely necessary to the technique and the conventions used.

3.3.6 Formal Z Specification Notation

Learning about Z consisted of reading books and manuals by Potter *et al.* [24], Ince [17], and Spivey [30] [31] and gaining some practical experience with research at HCSD. This technique required some mathematical expertise in set theory and predicate logic. Learning to use the commercially available tool for type-checking required some technical skill. Resolving type errors may not be trivial for practitioners with little mathematical expertise. My attempt to specify the system in Z from the *ad hoc* specification was difficult because there were no guidelines to systematically organise and translate the requirements to Z. This task was facilitated when the translation was derived from the threads-based specification. As with the Structured Analysis and threads-based specifications, the Z specification addressed only the functional requirements of the system.

3.3.7 Industrial Concerns on Specification

The third task in this research consisted of obtaining an industrial outlook on the use of specification techniques. Hence, interviews were sought with some practitioners of three of the techniques used in industry (*ad hoc* natural language approach, threads-based technique, Structured Analysis) [5] [21]. Common specification practice in industry, limitations of the techniques, and barriers to writing good specifications were some of the issues discussed. I also had contact with the Software Productivity Centre (SPC) [4] [25] in Vancouver, an organisation dedicated to fostering standards and best practices in software engineering. The SPC has offered a course on how to successfully specify software requirements [3].

3.4 Importance of Process During Requirements Specification

During the specification exercises, it became evident that knowledge of any type of specification technique is not necessarily sufficient in requirements specification. The gap between knowing a technique and deriving a system specification using the technique still needs to be bridged. This transition is facilitated by establishing a *process* to allow the systematic derivation of the specification document. A *process* is defined as "a series of activities over time that enable you to develop systems in a systematic manner" [5]. Without a process, starting a requirements specification can seem overwhelming and arduous. Establishing a process is even more crucial for large complex systems. In large systems, the number of specification writers at any one time during the software life cycle varies tremendously. At the beginning of the project, the size and complexity of the project and schedule constraints are such that almost all the engineers participate in the requirements analysis. Therefore, it is crucial that the process of specifying requirements be well established to facilitate the systematic and standard specification of requirements. As the project moves well into the design and implementation phases, a small team of engineers maintain the specification document.

3.5 Intrinsic Properties, Conventions, and Rules of Thumb

The more that specification techniques have conventions and guidelines on how to develop a specification, the easier it is to derive a specification. A specification technique is most conducive to systematically deriving a specification if it has numerous rules and conventions to guide the format and content of the various parts of the specification.

Unfortunately, what is lacking in most textbooks and literature about the various techniques is the distinction between the properties that are intrinsic to the technique and the properties that while extraneous, are helpful to produce a useful specification.

Usually, the more formal the specification notation, the better defined the intrinsic properties of the technique. In most of the literature, both kinds of properties are presented together. This deficiency in the literature has motivated the organisation of the chapters describing the four specification approaches. Thus, each of these chapters explicitly distinguishes between the intrinsic properties, conventions, and rules of thumb of each technique. Conventions are defined as basic procedures to be followed and accepted without exception. While these conventions are not necessary, they are very much part of the everyday practice of the methodology. These conventions are standards with which organisations ensure a level of uniformity in their application of the technique. On the other hand, rules of thumb are guidelines that have been established to assist in decision making during analysis and specification development. While rules of thumb are meant to be individual choices on how to apply a technique, they often help the specification be more readable and thus be more useful. Even if these rules of thumb are not followed, a specification may still be correct.

Since the intent of this research is also to assist those unfamiliar with various specification techniques, the chapters describing the techniques show how to develop specifications. More concretely, it is shown how the notations and/or conventions and/or rules of thumb of the techniques can be used to specify part of the SIS-Telereg system.

3.6 Combining Techniques

A combination of these techniques might be necessary to successfully analyze and specify requirements. There are two ways of combining specification techniques: selective application and sequencing. Selective application refers to using a combination of techniques whereby each technique is selectively applied to specify parts of the system for which it is best suited. Sequencing refers to using a technique to first obtain a preliminary version of the specification and then using some other technique to obtain a better and refined version of the specification. Sequencing may involve two or more techniques. Ideally, the preliminary specification document lends itself to being "translated" to the current specification using the technique.

The four approaches can be used together. Structured Analysis can provide a first cut at understanding the system requirements. Once a first level of understanding is reached, the requirements can be written in natural language as an intermediate specification document. From the natural language document, the requirements may be written in a threads-based style. If a formal specification is the ultimate target, the threads-based specification allows a smooth transition to the organisation of a Z requirements document. This sequencing may not be characteristic of the types of specification techniques (informal, semi-formal, formal) that are represented in this investigation. For instance, the threads-based approach may not lend itself to a formal specification technique other than Z or a notation with a similar document structure and specification style as the Z notation.

3.7 Other Observations

One advantage of the *ad hoc* approach is that it encourages the specification of requirements such as the system context, non-functional requirements, and validation criteria; none of the other three techniques do this.

It is quite conceivable that if the threads-based approach had been used exclusively, then I would not have uncovered information about the system which was uncovered using Structured Analysis.

Chapter 4

Ad hoc Informal Specification Approach

4.1 Introduction

The *ad hoc* informal approach is likely the predominant approach used by companies to specify systems. It is widely used because it requires little technical expertise, i.e. knowledge of special notations or tools, to specify requirements. However, this approach also presents numerous problems in its use of natural language as its notational tool. The natural language specification approach is often described in software engineering textbooks. The description in this chapter is based on Sommerville [29], Pressman [26], Parnas *et al.* [23], and standards ANSI/IEEE Std 830-1984 [32] and DOD-STD-2167A [11].

The natural language specification approach mandates the minimal constraints that the *ad hoc* natural language specification document must contain all the requirements necessary for a Requirements Specification. Consequently, the document is organised such that all the sections necessary for a complete Requirements Specification are present, and requirements relating to a certain section are specified in that section only. Each section is written in free form using natural language. Certain rules describe the expected content or scope of each section but there are few guidelines, except for standards such as DOD-STD-2167A and ANSI/IEEE Standard 830, on how the requirements are written. The most commonly used sections include an introduction, system context, system evolution, functional requirements, non-functional requirements, validation criteria, system data, traceability, and glossary [29] [26]. Any conventions used in this approach would be customised by a particular organisation to match its needs, and therefore, none are described in this chapter. Otherwise, the bulk of the contents of Chapter 2, on what a requirements document is and what its characteristics are, must be repeated here.

Section 4.2 defines and describes the *ad hoc* natural language approach. It also summarises the guidelines given in standards ANSI/IEEE Std 830-1984 and DOD-STD-2167A as related to system specification. Besides, Section 4.2 states how a natural language specification may be developed and the kind of tool support available. Further, the section hints at how to start an *ad hoc* specification and mentions tool support for this kind of specification. Section 4.3 gives the outline for the natural language specification of the SIS-Telereg system and gives some examples of requirements under each section of the document. Section 4.4 evaluates the natural language specification approach.

4.2 Natural Language Specification Approach

Natural language specifications are the most common kinds of specification documents produced by industries today. A natural language Requirements Specification consists of a series of requirements in natural language which are partitioned into sections based on the various kinds of requirements which the specification document should contain. The only intrinsic properties of the approach is that all requirements must be specified in a Requirements Specification. Adopting the military standard 2167A imposes constraints on the form and content of the specification. Any other constraints would be considered conventions of the approach. Depending on the size and complexity of the system and custom practice, conventions will vary between practitioners . Therefore, this section will describe only the kind of requirements that must be specified. Most software engineering textbooks and ANSI/IEEE Std 830-1984 offer guidelines on how to write requirements and provide prototype outlines for the document. These outlines are meant to be adapted to the organisations' needs (e.g. the names of the sections into which the document is organised can be changed to match the needs of each organisation).

Information conveyed by all the sections is necessary for a useful Requirements Specification. The most commonly used sections include an introduction, system context, functional requirements, system evolution, non-functional requirements, exceptional events requirements, validation criteria, system data, traceability, and glossary. Natural language is the notational tool used to specify the requirements. Overall, the Requirements Specification documents the desired behaviour of the system in terms of what is externally visible. It should contain everything needed for building the system and should not contain any design or implementation decisions. Any assumptions must be documented.

The introduction section gives an overview of the entire specification and describes the purpose of the specification as well as its intended audience. The introduction describes how the system is to be used and gives any background information to help understand how the system has changed and why a change is necessary. The benefits and goals of the proposed system and how it fits within the organisation are specified. A complete list of all documents referenced in the specification is identified. The introduction section also describes how the specification is organised.

The context section describes the relationship between the proposed system and its environment by defining the physical, hardware, peripheral, and software context of the system. If the system defines a product which is part of a bigger system, then the functions of the other components and their interfaces to the product are described. The physical context identifies the components which interact with the system and their interfaces to the system. For instance, components which are used to provide inputs to and receive outputs from the system are devices in the physical context. The peripheral and hardware context identify the machine and device support for the system. They describe the functions of the various devices and how they interrelate to one another. Diagrams and figures are used to show the interaction of the various components of the system and their interfaces. The software context describes the machines on which the software is to run.

The functional requirements section is the largest and most important part of the specification. It describes the desired behaviour of the system and states how inputs to the system are transformed into outputs. Validity checks on the input data, system state changes, system responses to both normal and error conditions, and system response in cases of system degradation are described. Error conditions include communication failure and error handling. Diagrams to show how the various software functions interrelate may be included. The section also describes any finite state machine behaviour that the system might have.

The non-functional requirements section describes constraints imposed on the operation of the software. Non-functional requirements include details related to design constraints and system performance. Design constraints imposed on the implementation of the system are stated and justified. Design constraints include the programming language in which to code the requirements, operating environment to be used, any mandatory standards that are to be followed, resource limits, hardware limitations, disk space, and memory requirements. If the system is to be implemented on special hardware, the hardware and its interfaces are described. Performance requirements describe the response and recovery times expected from various software components and the accuracies of outputs. Timing constraints such as how often an input occurs and how often each output should be computed and how accurate it should be, must be specified. This section may specify the number of terminals to be supported, the number of simultaneous users to be supported, size of tables and files, and so on. Mathematical calculations may also be specified in the non-functional requirements section.

The next three sections describe exceptional events, system evolution and validation criteria. The exceptional events section gives a description of what the system does if an undesired event occurs and a requirement cannot be fulfilled. The system evolution section describes anticipated changes to the hardware, software, and peripheral components. To the extent that each functional requirement is clearly identifiable, the validation criteria section describes how to demonstrate that each requirement is satisfied. Requirements may be tested by analysis, demonstration, or inspection.

The system data section describes the characteristics of data such as capacity, quantity, and composition. The traceability section states how each requirement is traceable to documents from previous stages of development and documents derived from the specification.

The last section usually includes a combination of glossary of technical terms used in the specification, table of contents, appendixes, and indexes to make the requirements document easier to use. References to all documents which relate to the software are documented in a bibliography section.

4.2.1 ANSI/IEEE Std 830-1984

The ANSI/IEEE Standard 830-1984 is a standard document prepared by the IEEE Societies to guide in the production of Requirements Specifications. Use of IEEE 830 is not mandatory; its purpose is to assist in the good practice of specification of software requirements. More specifically, it is aimed at helping in the development of outlines for Requirements Specifications as well as defining the form, content, and qualities of the documents.

4.2.2 DOD-STD-2167A

The military standard 2167A "establishes uniform requirements for software development that are applicable throughout the system life cycle". This standard mandates the use of "systematic and well documented software development methods" [11]. The standard establishes some guidelines on the outline, content and quality of the software and associated documentation. One important practical use of standard 2167A is that it defines a number of basic terms and an overall organisation that allows customers, contractors, and sub-contractors to communicate effectively at a management level. If the military standard 2167A is followed, then Data Item Descriptions (DIDs) are used "to describe a set of documents recording the information required by this standard" [11].

4.2.3 Natural Language Specification Development

A sensible way to start a natural language specification is to write the requirements in the order in which they appear in the Requirements Specification: the introduction, the system context, the functional requirements, and so on. If the specifier has a good understanding of the content of a Requirements Specification, the main challenge would be to acquire and understand the system requirements and allocating them to the appropriate section in the specification. Any conventions mandated by the organisation for which the specification is written must be followed.

4.2.4 Tool Support for Natural Language Specification

A valuable tool support for this approach is a computer-based word processor to manage the text file. Customised tools may be built to perform any kind of analysis on the specification. However, this is not an easy task because of the lack of structure in the way the requirements are written. Further, a tool which identifies, controls, tracks, and reports changes (e.g. red lines, strikeouts) aids the subsequent maintenance of the document.

4.3 Worked Example

Figures 4.2 and 4.3 give the outline for the natural language specification of the SIS-Telereg system. The specification starts with the introduction section which states the purpose of the document and how it is intended to be used. Additionally, the introduction section provides some background information on how the system has evolved over time and the rationale for these changes; it does not describe the advantages and disadvantages of the system.

The specification document then describes the external interfaces of the SIS-Telereg system with the input and output devices, the hardware and the peripherals in the context section. In the SIS-Telereg system, the touch-toned telephone and a business line enable students to provide inputs to and receive outputs from the system. The peripheral and hardware contexts describe the communication mechanisms between a user and the SIS-Telereg system. More specifically, it describes how touch tones entered by system users arc converted to a data stream that is understandable to a computer and how messages are composed at run time and relayed vocally to the system users. The machine support for the software is described in the software context section.

The next section of the document presents the functional requirements of the system, that is, how the system is expected to behave. The functional requirements form the bulk of the specification. Functional requirements are described in terms of the modes of operation (states) that the system may be in. The functions supported by the system during each mode of operation are specified. The operational constraints on the software are described in the non-functional requirements section. They document constraints

1 **Introduction**
  1.1 Purpose
  1.2 Background
  1.3 Sources and Acknowledgements
    1.3.1 Sources of Information
    1.3.2 Acknowledgements
  1.4 Organization

2 **Context**
  2.1 Physical Context
    2.1.1 Touch-tone Telephone
    2.1.2 Business Line
  2.2 Peripheral and Hardware Contexts
    2.2.1 VOCOM-40 Voice Processor
    2.2.2 IBM 390 Mainframe
    2.2.3 Protocol Converter
  2.3 Software Context

3 **Functional Requirements**
  3.1 Logging In
    3.1.1 Validating Identification
    3.1.2 Withholding Student Registration
    3.1.3 Checking for Outstanding Fee Payment
    3.1.4 Exiting the "Logging In" Mode of Operation
  ...

Figure 4.2: Table of contents, *ad hoc* natural language specification

...
3.2 Preprocessing
  3.2.1 Computing the Current Fee Assessment
  3.2.2 Checking the Access Dates
  3.2.3 Checking for Eligibility to Register
  3.2.4 Exiting the "Preprocessing" Mode of Operation
3.3 ...

4 **Non-Functional Requirements**
4.1 Product Requirements
4.2 Response Time
4.3 Reliability
4.4 Security
4.5 Concurrency

5 **Exceptional Events**

6 **System Evolution**
6.1 Software Evolution
6.2 Peripheral Evolution

7 **Validation Criteria**

8 **Glossary**

Figure 4.3: Table of contents (cont.), *ad hoc* natural language specification

related to response time, reliability of the system, security and concurrency. The exceptional events section describe the behaviour of the SIS-Telereg system if hardware and software failures occur. Further, it describes how the software must attempt to recover gracefully and what can be done to minimise down times during system failures. The system evolution section states how the functionality of the system is predicted to change. It also describes how the existing machine configuration can be enhanced by the addition of a peripheral device. The set of acceptance tests for the system and a glossary of technical terms in the application domain are found in the last two sections of the document.

4.4 Strengths and Weaknesses of the Natural Language Specification Approach

This section presents an evaluation of the natural language specification approach.

4.4.1 Strengths

1. **The *ad hoc* natural language approach does not require much technical skill.**

   This approach requires the least amount of skill compared to semi-formal and formal approaches to specification. It is widely described in literature; the approach is described by most software engineering textbooks and several international standards. No special expertise is needed except an understanding of what the content and scope of each section of a requirements document must be.

2. **An *ad hoc* natural language specification is readable and accessible.**

   Since natural language is used to convey the requirements, the specification can be accessible and readable to customers, design engineers, test engineers, and maintenance personnel.

4.4.2 Weaknesses

1. **An *ad hoc* natural language specification is subject to ambiguity.**
   What enables the natural language approach to be widely used also causes most of the disadvantages of the approach. Overall, writing requirements in natural language is *ad hoc* and open ended. This approach is subject to ambiguity and imprecision because these properties are inherent in the "notational tool" (natural language).

2. **The level of ambiguity in an *ad hoc* natural language specification may vary for no justifiable reason.**
   Using natural language can potentially result in expressing requirements at different levels of detail. When using natural language, it is also difficult to clearly distinguish between functional requirements, non-functional requirements and design constraints.

3. **The *ad hoc* natural language approach does not facilitate the specification process (i.e. little effective guidance to specifiers).**
   Other than the guidelines on how to organise a specification document, there are no guidelines on how to organise requirements within each section. Each company has its own conventions. There are no guidelines to maintain consistency of style. There is no systematic way of writing requirements; therefore requirements are not all specified in a uniform manner. In a large software project, this technique can result in various specification styles throughout the specification document since there are very few helpful guidelines to assist the various specifiers in their efforts.

This may make the tasks of the specification users, namely the design and test engineers, more difficult.

4. **An *ad hoc* natural language specification is not easily maintainable.**
   An *ad hoc* natural language specification document is not easily maintainable. The approach offers no guidelines to ensure that requirements related to a particular concern are isolated to one place. Therefore, it may be time consuming to localise all places where revisions need to be made.

5. **The *ad hoc* natural language approach does not facilitate the validation of requirements.**
   The informal, narrative description of requirements makes it hard to assess whether the information is factually correct. An expert knowledge and careful study is required to ensure that there are no ambiguities, errors, inconsistencies or omissions in the specification. The *ad hoc* free style writing makes it hard to validate the requirements because there is no way of ensuring that any inputs to a process results in an output which is sent to the proper destination. Also, with the free style, there is no way of ensuring that data integrity, consistency and accuracy are maintained.

6. **The natural language informal approach does not facilitate tool support.**
   It is difficult to build tools to perform any kind of parsing or analysis on a natural language specification. This is because any parser will have to be very "smart" to get useful information from an *ad hoc* document. There are no tools to check the consistency of a natural language specification. Review teams are set up to analyse the specification for consistency and ambiguity. Any type of analysis such as test case generation is a labour intensive task. Natural language understanding tools may be used to parse a natural language specification for analysis but the

results might not have a high level of success because of the lack of structure in the specification document.

4.5 Conclusion

This section presents some observations on the natural language specification approach.

4.5.1 Remarks About the Natural Language Specification Approach

The natural language specification approach is the most widely used approach because it is the simplest approach to use compared to CASE methodologies and formal techniques. It does not require much technical expertise since natural language is the notational tool used in this approach. However, the flexibility of style allowed by the approach is disadvantageous in that it becomes difficult to build any kinds of tools to do analysis on the specification and even to analyse the specification manually.

4.5.2 Evaluation of *Ad Hoc* Informal Specification Approach

While the *ad hoc* informal specification approach may be the simplest and most widely used specification approach, it also poses numerous problems. The use of natural language in a free writing style allows great flexibility in specifying requirements but also results in ambiguity, imprecision, incompleteness, and non-uniformity.

Chapter 5

Semi-Formal Specification Technique

5.1 Introduction

Of the semi-formal specification techniques, Structured Analysis is probably the most popular and widely used. Structured Analysis has been used for almost two decades now [12]; it is relatively mature and is supported by a number of established commercial tools. Structured Analysis emphasises the functional and control views of a system. The Structured Analysis approach described in this chapter is based on DeMarco [10] and Hatley/Pirbhai [15].

Structured Analysis (SA) is a systems analysis technique that was designed to build and maintain structured specifications. The technique was initially intended for the representation of the data processing parts of systems. When reactive systems with complex control structures became prominent, Structured Analysis was found lacking in that it could not depict the finite state machine behaviour of these systems. Extensions (SA/RT) were thus integrated in the technique to cater for the control nature of systems.

Structured Analysis is a methodology that describes a system as a function from inputs to outputs. The methodology has a well-defined syntax and a loosely defined semantics. Structured Analysis offers a graphical data flow model of a system and uses the divide and conquer principle to depict a hierarchical high level view of the system. Consequently, a Structured Analysis model consists of various levels of abstraction with each level placing emphasis on transformation of data. Each level of abstraction is called a data flow diagram and it depicts the functional decomposition of a system from the point of view of data rather than from the viewpoint of one user or group of users. A data flow diagram is essentially a picture showing the flow of information and how the information is processed by transformation centres called processes. The methodology does not have well defined semantics and therefore its use varies between practitioners as they each advocate various conventions and rules of thumb. Over the years, various commercial CASE tools for Structured Analysis have become available such as *CADRE Teamwork SA/RT*. Teamwork supports the DeMarco [10] and Hatley/Pirbhai [15] approach to SA/RT.

Sections 5.2 and 5.3 respectively describe the Structured Analysis methodology as originally invented and the extensions that were later added for reactive systems. The conventions and rules of thumb adopted by practitioners are also described. In addition, these sections describe tool support for SA and its RT extensions as well as how to develop an SA model. Section 5.4 illustrates how the principles of Structured Analysis may be applied to specify parts of the SIS-Telereg system. Section 5.5 summarises the difficulties in learning SA based on personal hands-on experience with the methodology and experience in instructing undergraduate students in its usage. The various difficulties that the novice user can encounter are also highlighted. Section 5.6 describes the strengths and weaknesses of Structured Analysis. Finally, Section 5.7 concludes with some observations about the methodology as a semi-formal specification technique.

5.2 Structured Analysis (Pure)

Structured Analysis has been in use since the early 1970s [15] and is ideally suited for the specification of data processing systems. Structured Analysis, as originally intended, is a methodology that describes a system as a function from inputs to outputs. Structured

Analysis offers a graphical data flow model of a system and uses the divide and conquer principle to depict a hierarchical view of the system. Structured Analysis has a well-defined syntax and a loosely defined semantics. Structured Analysis uses notational tools such as Data Flow Diagrams, Data Dictionary, and Process Specification, and the resulting product is an SA specification document.

Section 5.2.1 describes the notation used in Structured Analysis. The two basic types of syntactic correctness checks in SA, namely levelling and balancing, are described in Section 5.2.2. Sections 5.2.3 and 5.2.4 present the conventions and rules of thumb for SA respectively. The development of an SA model is specified in Section 5.2.5. Finally, Section 5.2.6 describes tool support for SA.

5.2.1 Notation Used in Structured Analysis

Data Flow Diagrams

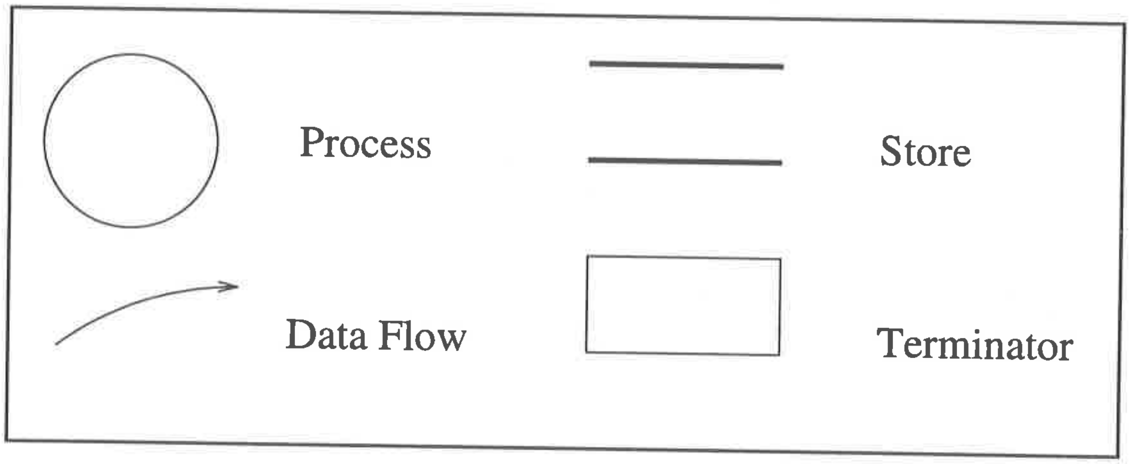


Figure 5.4: Elements of a Data Flow Diagram

Data Flow Diagrams (DFDs) depict the functional decomposition of a system from the point of view of data rather than from the viewpoint of one user or a group of users.

In other words, DFDs are pictures that show how data flows through a system and how the data is changed. Figure 5.4 shows the elements of a DFD.

A data flow, depicted by a directed arc, is a data path containing information of known composition. A data flow serves as a link between elements of a DFD. The direction of the arrow indicates the flow of data. A data flow can also be double-headed to indicate that the same kind of data flows in both directions. A data flow must be connected to other elements of a DFD. A data flow must originate from a process, terminator, or store and must end in a process, terminator, or store. Every data flow must have a name that describes its contents.

A process, depicted as a bubble (thus, sometimes called a "bubble"), is the basic element into which a system is decomposed. A process specifies the transformation of input data flows into output data flows. Therefore, a process is drawn with data flows going into it and data flows leaving it. Every process must have a descriptive name which should hint at its function. A process must build its outputs using only information from the data flows explicitly shown flowing into it and constant information (data store). Conversely, any data flows shown as input to a process must be used by that process.

Terminators indicate the external agents interacting with a system and they can either be originators or receivers of data. Hence, terminators provide the net inputs to a system and receive the net outputs from the system. Terminators are specified with rectangles. Terminators must also be named.

A store is a temporary repository of data and is denoted with two parallel horizontal lines. The contents of a store remain unchanged until they are replaced with new data. Any process may repeatedly use the contents of a store. Stores are named after the data they contain. When data flows are drawn to or from a store, they need not be named since their contents are exactly those of the store, and therefore the same name as the store would be used for these data flows. Data must enter and leave all stores. Only the net direction of flow to and from a store must be shown. Double headed arrows are drawn between a process and a store when the process needs to update the file and the information from the file is to be used for some purpose other than the update.

Therefore, Data Flow Diagrams consist of processes that are interconnected by means of data flows to other processes, terminators or stores. The data flows are meant to represent the set of possible data paths and are therefore not used to prompt a process to start work. Bi-directional data flows may only occur when they are connected to a data store. Data Flow Diagrams must be acyclic in that data flows cannot return to their starting points.

Data Dictionary

A Data Dictionary (DD) is a document that contains, in alphabetical order, definitions of terms used in a Structured Analysis specification document. It must include descriptions of data flows and stores. A Data Dictionary is an essential part of a specification document.

Definitions in a DD must be accessible by unique names. It must be a document that can easily be updated and maintained. Definitions of Data Dictionary Entries (DDEs) must be adequately captured by the names of the entries. Comments may be used in the DD for any clarification purposes and they are denoted by text enclosed in asterisks.

A DD documents the composition of its entries and describes what the components (data elements) are and how they are related. Definitions must cover all possible instances of an entry. Definitions in the DD are partitioned top-down with the bottom level components being the basic elements of the DD. In other words, the DDEs at the bottom layer are not defined because they are the basic "types" of the DD. The notation "=" is used to associate a data item with its definition.

A data flow can be a single item, a composition of two or more items whereby each component item is necessary to make up the data flow (conjunction), or any one of a list of items (disjunction). In the former case, the different components are composed with a plus sign (+); the order in which the components appear is insignificant. In the latter case, the various choices are listed within square brackets ([]) and separated by a vertical bar (|). For example, a student identification can consist of a student number and a birth date. Its corresponding entry in a data dictionary could be as follows:

```
student_id = student_number + birth_date
```

If a student can either request the addition or deletion of a course from her registration schedule, an entry to illustrate this could be:

```
register_request = [ add_course | drop_course ]
```

If a data element is optional, it is enclosed within parentheses. Consider the data dictionary entry for a student name consisting of a last name, a first name and an optional middle name:

```
student_name = last_name + first_name + (middle_name)
```

Iterations of a component is indicated with curly braces ({ }), e.g. the DDE

```
student_number = 1{digit}8
```

indicates that a student number consists of 8 digits. Combinations of [], |, {} and () can be used together to correctly describe a data flow.

Different names are possible for the same data flow because the same data can be known by different names by different users. Aliases are used to indicate a previously defined data item. For example, in the SIS-Telereg system, "dept_register_requests" and "student_register_requests" are both aliases for "register_requests". Two different names were used to show that the requests came from two different users, namely a university department and a student and, while both the requests contained the same information, their format were different.

If there are self-defining or "basic" terms in the data dictionary, they are not defined further. For instance, in

```
student_profile = sex + age
```

provided "sex" and "age" are not used to stand for more than their well-known standard definitions, there is no point in defining these terms since their meanings are obvious.

If a data item has a known non-numeric or literal value, it is enclosed in double quotes. For instance, if the data flow "schedule_type" can either take the value of "REGULAR" or "STANDARD_TIMETABLE", its entry would be:

```
schedule_type = [ ``REGULAR'' | ``STANDARD_TIMETABLE'']
```

In addition to defining a data item in a Data Dictionary, the attributes of the data item must also be described. For instance, a data item class must be provided; in other words, a DD entry needs to indicate whether the entry is a data flow or a store. A data flow must be further detailed to indicate whether it is a data element of a data flow or an alias.

Data Dictionary Entries must be cross-checked to ensure that they are self-consistent. All data flows must be defined in the DD and the definitions for their components and sub-components must be consistent with each other.

Process Specifications

A Process Specification or PSpec is a document that gives a concise and narrative description of a process. A PSpec describes the transformation rules that generate the output signals of a process from its input signals. A PSpec describes the contents and

```
P-Spec 3: Withdraw money

INPUT/OUTPUT:
balance: input
withdrawal: input
balance_message: output
overdrawn_message: output

BODY:
IF withdrawal <= balance THEN
    balance = balance - withdrawal
    give balance_message
ELSE
    give overdrawn_message
```

Figure 5.5: Example of a Process Specification

processing of data items and how their values are established. A PSpec only describes what must be done to the data and not how this is to be done. Comments may be added to help the reader understand the purpose of the PSpec. Figure 5.5 is an example of a PSpec.

Mathematical formulae should be used whenever possible to express the transformation rules of a process or parts of a process. Graphical illustrations such as tables, diagrams, and graphs should also be used as much as possible since they increase readability and have the potential to decrease ambiguity.

DeMarco [10] mentions several alternatives for process specification, namely Structured English, decision table, combination of decision table with Structured English, and decision tree which is basically a graphical representation of a decision table.

Structured English

Structured English (SE) is a restricted subset of the English language. Structured English combines the rigour of a structured programming language with the readability of English. It allows limited vocabulary and syntax. The vocabulary consists only of verbs (meaningful ones unlike "process" or "handle"), terms defined in the Data Dictionary, and certain reserved words such as CASE, IF...THEN...OTHERWISE, REPEAT...UNTIL, and relational attributes such as EQUAL, AND, OR and so on. The syntax of SE consists of simple sentence structure in the imperative mood and may not contain adjectives and adverbs. There are no punctuation rules. The goal of using Structured English is conciseness, precision, and readability while still in a natural language framework. The PSpec illustrated in Figure 5.5 is written in Structured English.

However, a PSpec written with the utmost care in SE might look like code to a user because of its limited syntax and vocabulary. In that case, appropriate words and phrases may be inserted at suitable places to make the description easier to read and understand. Capitalization of reserved words and hyphenation as used in the convention for data names may be removed. Indentation may be adjusted to increase readability. In other words, the PSpec should be rewritten so that it is in a more narrative style while retaining a somewhat restricted syntax.

Decision Table

Decision Tables (DTs) are used when there are a considerable number of policies to be chosen from and selection of the appropriate one is dependent on a number of conditions. DTs can also be used to evaluate policies for completeness and consistency. Figure 5.6 is an example of a DT.

DTs may also be used in combination with Structured English to describe a process.

	Rules			
	1	2	3	4
Conditions				
C1	Y	N	Y	N
C2	Y	Y	N	N
Actions				
A1	N	Y	Y	Y
A2	N	Y	Y	Y
A3	N	Y	N	N

Figure 5.6: Example of a Decision Table

In the example of Figure 5.6, actions A1, A2, and A3 could be described in SE.

Decision Trees may also be used in lieu of Decision Tables because they are merely graphical representations of DTs. Decision Trees are used if they are more readable than Decision Tables to a user.

5.2.2 Levelling and Balancing

Levelling and balancing are two kinds of syntactic correctness checks that are basic to Structured Analysis.

Levelling

Levelling is the partitioning of a system into subsystems such that a system can be decomposed in a top-down hierarchical fashion so that there is a smooth transition from most abstract to most detailed. Thus, a process can potentially be decomposed into a child DFD that details the processing of the parent process. An SA model therefore consists of a top level, called the Context Diagram (CD), one or more levels of Data Flow Diagrams, and a bottom level of functional primitives, that is, processes that are not further decomposed.

The Context Diagram is a special case of a DFD. The Context Diagram, being the highest level of a Structured Analysis model, depicts the system and its interaction with its environment, and it therefore presents the most abstract view of a system requirements. The Context Diagram delineates the scope of a system. A Context Diagram consists only of one process representing the system itself, terminators to show the external agents interacting with the system, and data flows linking the terminators and the system bubble. The Context Diagram shows the net inputs and outputs of the system. An exception arises in the naming of processes in the Context Diagram. Because the single process is meant to represent the system, it is given the name of the system. Stores may not appear on the Context Diagram.

The intermediate levels of Data Flow Diagrams make the bulk of the system. They contain the usual elements of a Data Flow Diagram. Each DFD is a refinement of a parent process and presents in more details the purposes of the parent process. Off-page data flows (data flows that do not originate from any source or do not end in any destination) may occur on any child data flow diagram except on the Context Diagram. However, off-page data flows actually have sources and destinations from parent data flows which have sources and destinations.

Functional primitives are processes that are not further decomposed into child data flow diagrams and therefore make up the lowest level of data flow diagrams. Since they are not further decomposed, their processing are described in Process Specifications (PSpecs) so that levelling is maintained. There must be a PSpec for each functional primitive. Functional primitives may occur at higher levels of DFDs, other than the top level, for those processes that need no further refinement. Therefore, PSpecs may appear at any level of decomposition but they will definitely occur at the lowest level of decomposition.

Hence, a properly levelled SA model consists of a Context Diagram at the top level, several levels of DFDs, and functional primitives whereby a PSpec is associated with each primitive. Each DFD may only exist because it is a refinement of a parent process at a previous level. Two or more processes cannot share a child DFD. Every bubble must have a child DFD or a PSpec. The Context Diagram can only show data flows, terminators, and a single bubble. The inputs at the context level should be sufficient for the generation of its outputs. Terminators may not be found at other levels other than the context level. There should be one PSpec per primitive process. A PSpec must not be written for a process that is further decomposed.

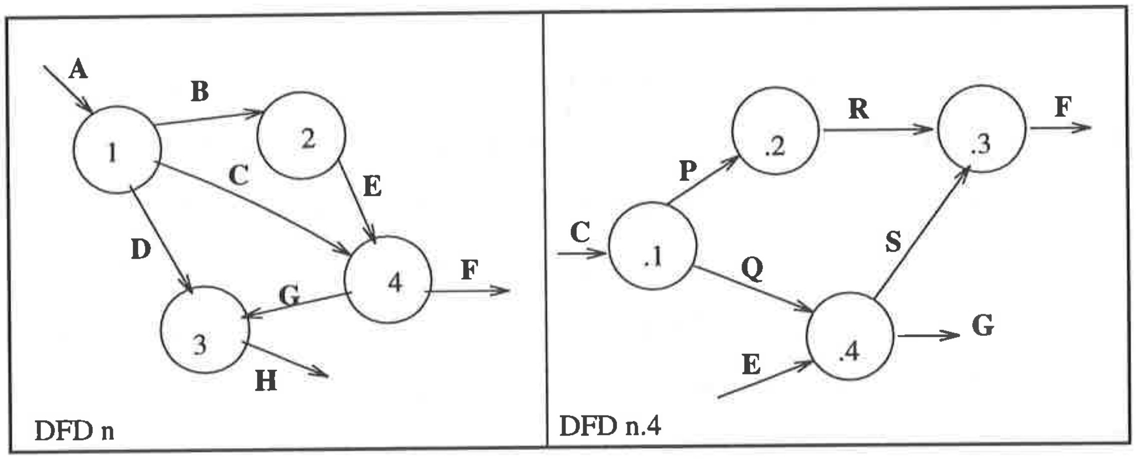


Figure 5.7: Example 1: concept of balancing

Balancing

Balancing is a syntactic correctness check which involves the Data Dictionary. Balancing is the relationship that exists between parent and child diagrams in a properly levelled model. Since a DFD and its parent process represent the same information at different levels of details, their inputs and outputs must contain the same information. Therefore, balancing is the concept that data flows going in and out of a parent process must also be the net input and output data flows respectively in the child Data Flow Diagram. In essence, whatever flows in and out of a parent process must also be inherited by the child DFD. The processes in the child DFD need not each inherit all the inputs and outputs of the parent process. Balancing should also apply to a PSpec because the latter is the child of the process it describes. Thus, a PSpec and its primitive process must share the same inputs and outputs. In fact, balancing provides the main self-consistency check of an SA requirement model.

The concept of balancing follows from the idea of levelling. For instance, Figure 5.7 is balanced. Process 4 in DFD n was further decomposed into child DFD n.4. As can be seen, the input data flows **C** and **E** and the output data flows **F** and **G** occur in both the child DFD and in the parent process. But data flows **P, Q, R,** and **S** occur only on DFD n.4 and not on process 4 on DFD n. This, however, does not violate balancing because these data flows are internal to the processes in the child DFD n.4 and therefore do not contribute to the net inputs and outputs of the DFD.

Frequently, composite data flows are decomposed as they pass from a parent process to a child diagram. With the possibility of expressing data items in terms of conjunctions of data elements and disjunctions of alternative data items, balancing is more than just matching input data flows and output data flows attached to a process to those of the child DFD. It is necessary to check against the definitions of the data flows in the Data

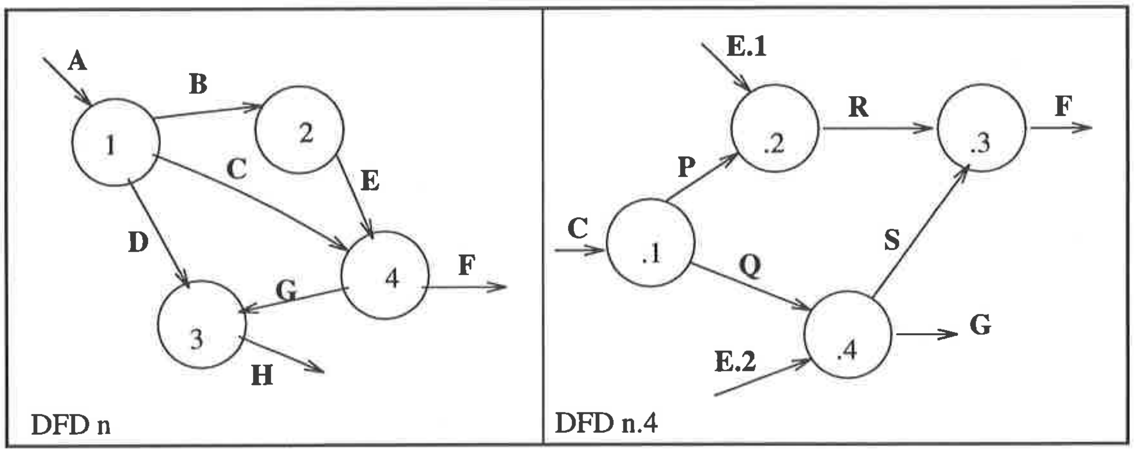


Figure 5.8: Example 2: concept of balancing

Dictionary,

If a data flow is a combination of component data items and the data flow is used at the parent level, then either the data flow itself has to be used in the child DFD or all the components of the data flow have to be used in the child DFD in one manner or another regardless of which bubbles they are attached to. A single data component may be used at one or more processes in the child DFD. In Figure 5.8, if data flow **E** has been defined as

```
E = E.1 + E.2
```

then both data elements need to occur in the child DFD, as illustrated in Figure 5.8. Data flow **E.2** could also be drawn as input to another process as well, say, process .3.

However, if a data flow is a disjunction of data items and it occurs at the parent process, then, at the child DFD, any one or more of the alternative data items or the data flow itself would be needed to balance the data flows. Therefore, suppose the Data Dictionary entry for data flow **E** is as follows:

```
E = [ E.1 | E.2 ]
```

Then Figure 5.8 would still be balanced. Removing either of data flows **E.1** or **E.2** would not cause Figure 5.8 to be unbalanced. Figure 5.7 would also be balanced under this definition of **E** in the Data Dictionary although, in this case, it might then possibly not make sense to have split the data flow into components.

If data flow **E.1** is further decomposed so that its DDE is

```
E.1 = E.1.a + E.1.b
```

then, from the above definition of **E**, whenever **E.1** is the option to be used, both its components need to be shown on the child DFD attached to any processes. If **E.2** is the option to be used for **E**, then **E.1.a** and **E.1.b** need not be shown at the child DFD but **E.2** must be shown at the child DFD.

If instead data flow **E.1** is decomposed so that its DDE is now

```
E.1 = E.1.a + E.1.b + (E.1.c)
```

then the above description which holds true for the previous definition of **E.1** must also be true here. Also, whenever **E.1** is chosen to stand for **E**, data flow **E.1.c** may be optionally shown on the child DFD.

5.2.3 Conventions for Structured Analysis

Naming Rules For Elements in Data Flow Diagrams

Certain rules are followed in naming the elements of a DFD to improve its readability. If names are longer than a word long, they are connected with underscores. Since a data flow contains data of known composition, its name is a noun that adequately and precisely describes its contents. For example, if a data flow not only contains a student identification but also one that has been validated, the data flow should be named "valid_student_id" rather than "student_id" or just "identification".

A process is named with a strong action verb and a single noun. The verb describes the action done to the data flows and the noun describes the specific data on which processing is done. The bubble uses the input data flows to produce the output data flows; hence an action is done on the input data flows. Also, we know exactly what data flows are being used, so we know the specific object on which the action is being done.

Process names should be checked against their inputs and outputs. Consider a process whose function is to check whether an inputted student identification is valid and then either passes the valid identification on to another process or outputs a rejection message. The process name "Validate student identification" is more appropriate than "Process student" whereby the verb "process" is too vague and the object "student" is not as precise as "student identification".

Terminators and stores are named with nouns that appropriately describe them. They follow the same rules as those used for naming data flows.

Numbering System

The numbering system associated with levelling and numbering of processes is such that the model is self-indexing. The parent-child relationship is intrinsic in the numbering convention of processes. Various conventions are used and the one described here is based on the implementation offered by the CASE tool Teamwork. If any process is decomposed, the child DFD will have a level number equal to the number of the parent process. Each bubble is numbered, starting with zero for the one process at the Context Diagram level. In the first level of decomposition (level 0 since the parent process is numbered 0), the processes are numbered starting from 1 onwards in unit increments. In the next DFD levels, the processes are numbered starting with the decimal value .1 onwards in increments of 0.1 attached to a prefix equivalent to the level number at which the process appears. For instance, the child diagram of process .6 "Execute The

Registration Request" at level 3 is labelled "DFD 3.6 Execute The Registration Request" and its processes are numbered 3.6.1, 3.6.2, 3.6.3, and so on. In practice, however, the processes are simply labelled .1, .2, .3, and so on; it is understood that they are prefixed by the DFD number.

The numbering system for both DFDs and PSpecs are illustrated in Figures 5.9 and 5.10 respectively. PSpecs usually take on the name of the processes that they describe. The title of a child DFD or a PSpec must match the title of the parent process.

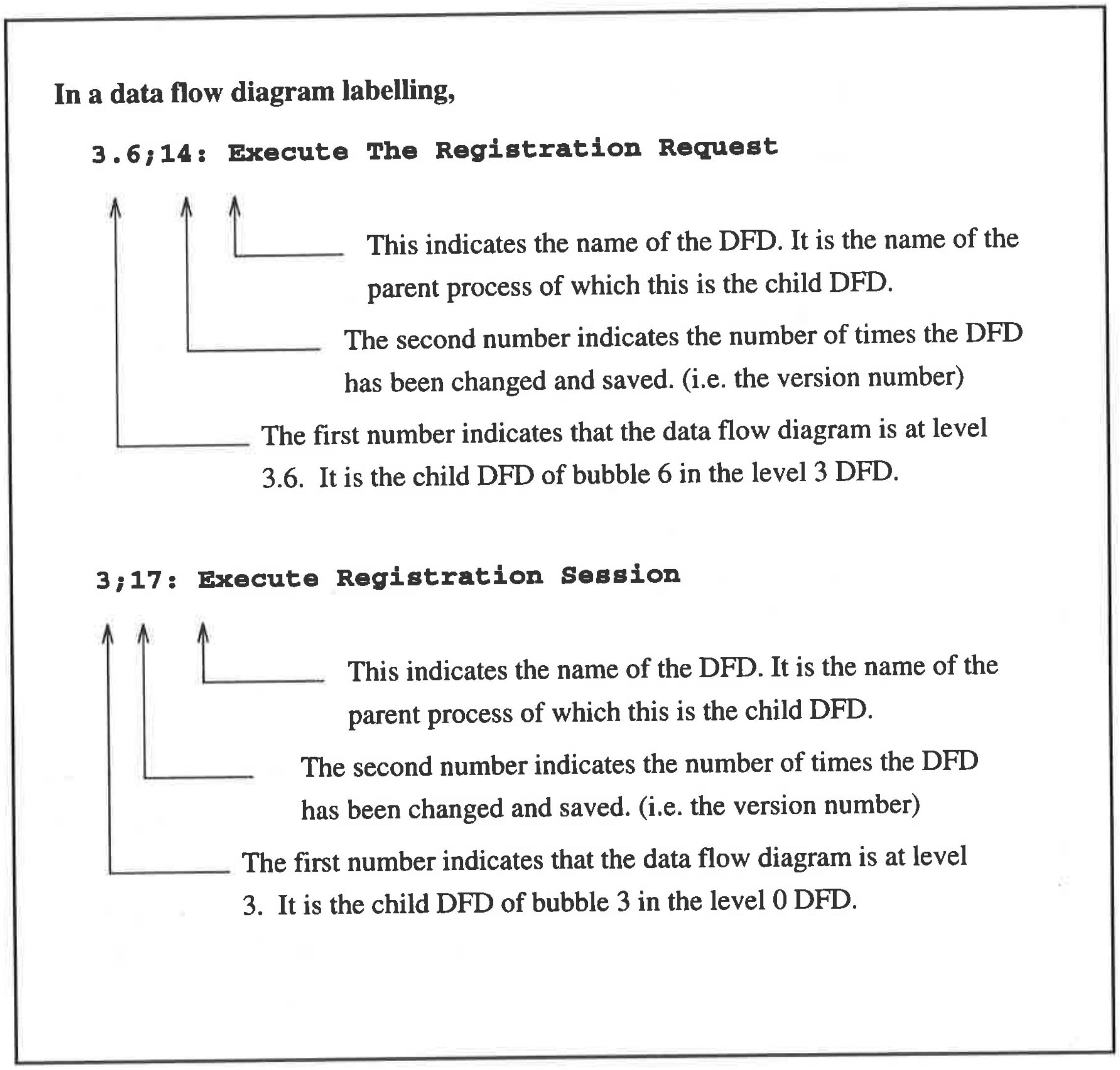
5.2.4 Rules of Thumb for Structured Analysis

In general, there should be seven ± two processes in a data flow diagram. An inexperienced user of the methodology might be tempted to stay within this range at all cost; however if diagrams are clearer with more than seven processes, then the range should be overridden. The branching of flows in the DFD can be a good indication of whether bubbles are too crowded.

An SA model should be useful in that the names of its data flows, processes, terminators, and stores should be descriptive of the functionality of the system. It should be checked whether the model is hard to read and understand. The analyst should aim to have an even partitioning of the system as well as to maintain the number and clarity of data flows in and out of bubbles.

Context Diagram

If there are many data flows from one terminator to the system bubble in the context diagram, they should not all be shown separately on the context diagram. Data flows should be grouped in a logical fashion so that the clutter of data flows does not make reading and understanding the system harder. The combination of data flows should then be named appropriately to capture the contents of its components.



In a data flow diagram labelling,

`3.6;14: Execute The Registration Request`

This indicates the name of the DFD. It is the name of the parent process of which this is the child DFD.

The second number indicates the number of times the DFD has been changed and saved. (i.e. the version number)

The first number indicates that the data flow diagram is at level 3.6. It is the child DFD of bubble 6 in the level 3 DFD.

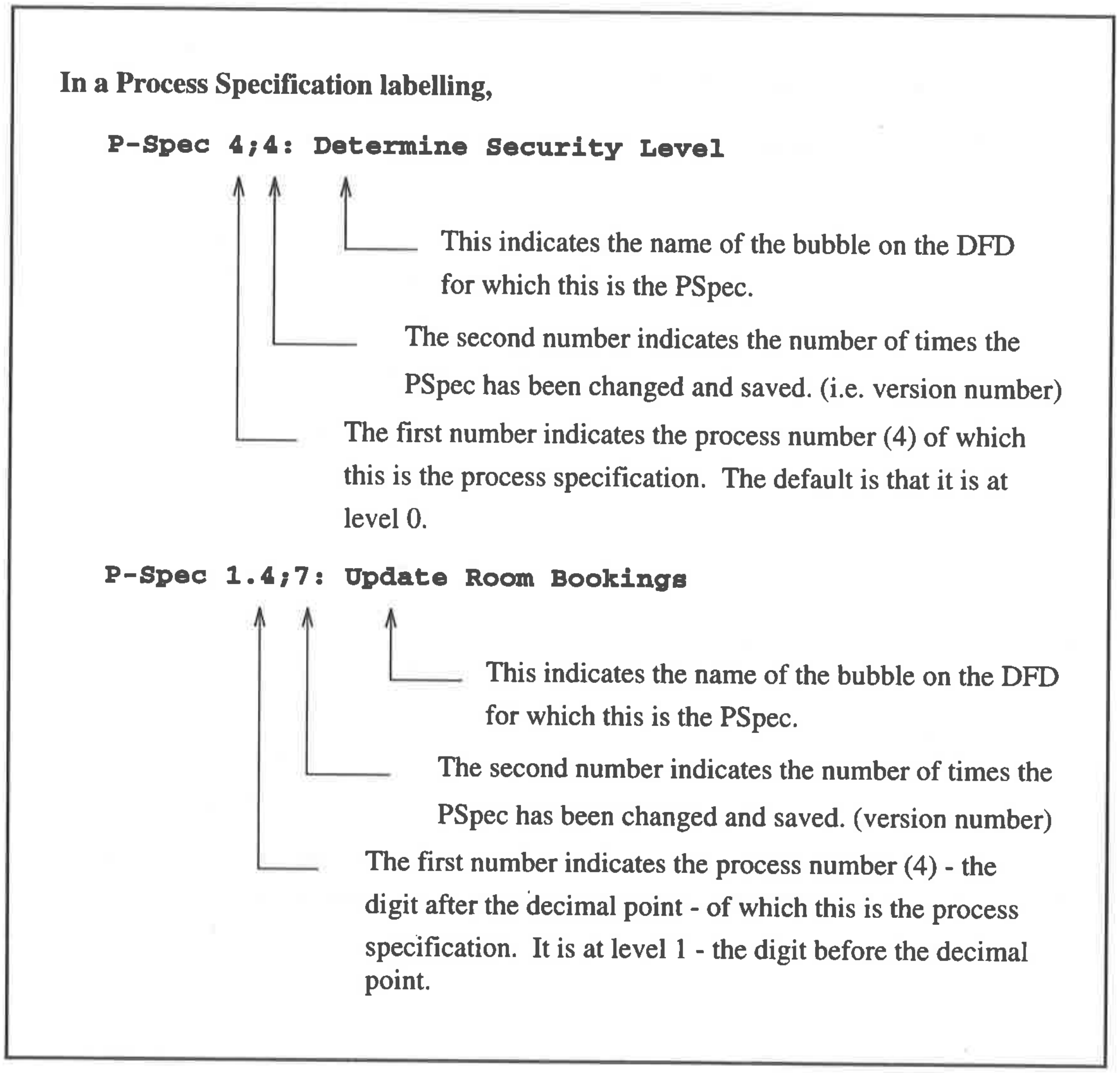
`3;17: Execute Registration Session`

This indicates the name of the DFD. It is the name of the parent process of which this is the child DFD.

The second number indicates the number of times the DFD has been changed and saved. (i.e. the version number)

The first number indicates that the data flow diagram is at level 3. It is the child DFD of bubble 3 in the level 0 DFD.

Figure 5.9: Data Flow Diagram labelling



In a Process Specification labelling,

`P-Spec 4;4: Determine Security Level`

This indicates the name of the bubble on the DFD for which this is the PSpec.

The second number indicates the number of times the PSpec has been changed and saved. (i.e. version number)

The first number indicates the process number (4) of which this is the process specification. The default is that it is at level 0.

`P-Spec 1.4;7: Update Room Bookings`

This indicates the name of the bubble on the DFD for which this is the PSpec.

The second number indicates the number of times the PSpec has been changed and saved. (version number)

The first number indicates the process number (4) - the digit after the decimal point - of which this is the process specification. It is at level 1 - the digit before the decimal point.

Figure 5.10: Process Specification labelling

The Context Diagram should be drawn so that data flows are given descriptive names and they are organized in functionally meaningful groups. A reader can be expected to list at least three functions of the system after briefly examining the context diagram.

Naming of Data Flows

Names for data flows should be chosen to represent not only the data which are contained but also what is known of the data. Vague names such as "input", "output","information" and "data" should be avoided since they add no value to the specification effort. Coded names like the ones used in programs should be avoided. For instance, using "INVALID-ERR" or "ERROR5" as a name for a data flow is not as meaningful as "invalid_id_msg" which is a message indicating that the identification is invalid.

If a data flow cannot be named, then it is either unnecessary or partitioning was poor. No two data flows can have the same name. Items must not be grouped into a data flow if they have nothing in common. Data flows names never contain verbs; they contain nouns only. If at all possible, crossed data flow lines should be avoided on a data flow diagram because they produce diagrams which are hard to understand.

Naming of processes

Each process should be named with an accurate descriptive name. The right choice of names for processes can possibly affect whether a model is meaningful and useful.

Bubble names should be tested against inputs and outputs. Vague names such as "process" and "handle" should be avoided when naming a process. If these words are needed, then the process is deemed an unnameable process and repartitioning may be needed. If two verbs are required when naming processes, then it is likely that more partitioning is needed.

Stores

A store is shown on a data flow diagram at the first level where it is used as an interface between two processes. Stores which are only relevant to the inside of some processes are concealed. All references are shown to a store at the first level where it occurs.

Process Specifications

Ideally, the size of a PSpec should be less than a page. A PSpec is not useful if more clarification is needed after one reads over it. PSpecs will most probably describe processes with more than just one single input and output data flow. Otherwise, decomposition of processes would be done at a level of detail which is not needed.

5.2.5 SA Development

Using Structured Analysis to model a system is an iterative process. A good model of a system cannot be obtained at a first attempt. Hence, one should be prepared to start over and to abandon earlier versions of Data Flow Diagrams so that they can be replaced with improved versions. As a model develops and as the requirements of the system change or become clearer, reconsidering how the system is being partitioned might be necessary.

Usually, the higher level Data Flow Diagrams will be more cluttered than the lower levels. In the SIS-Telereg system, however, this is not the case since the focus is only on a subset of the SIS system. There is no single correct model for a given set of requirements. Different people may partition the requirements differently and may come up with different models, each of which might be equally effective. While Structured Analysis is intended to be a methodology to analyse and model the requirements of a system, it is also sometimes used by some practitioners as a high level design documentation. Then, the design introduced in the model by partitioning must match the actual design of the system.

More specifically, to start a model, the context is established by identifying the net inputs and output data flows to the system. The data flows may then be used at any level by either working upwards or downwards. Interface data flows are labelled and processes are not named at first. After their interfaces are established, processes can be named based on their data flows. Error paths should be ignored at the first few attempts unless they affect the partitioning of the system.

5.2.6 Tool Support for Structured Analysis

Various CASE tools which provide support for Structured Analysis are available commercially. These tools can automatically validate an SA model with respect to the intrinsic principles of SA as well as some conventions of SA. *CADRE Teamwork SA/RT* is one such tool. The tool support described in this section is based on version 4.1 of Teamwork SA/RT [2]. Teamwork supports DFD syntax checking, balancing of processes, DDE expansion checking, PSpec checking as well as labelling and numbering of DFDs, processes and PSpecs.

DFD Syntax

A syntax check searches for violations of syntax rules in the DFD. A user may specify to have only the selected DFD checked or the selected DFD and its subtree checked.

Balancing of processes

This includes balancing of DFDs and PSpecs. The balancing of child DFDs and parent processes is checked based on the rules described in Section 5.2.2. A user may specify to have only the selected DFD checked or the selected DFD and its subtree checked.

Levelling

Since the levelling of data flow diagrams is automatically provided by Teamwork SA/RT, no checks are done on numbering and labelling of levels.

DDE expansion

During a balancing check, the DDEs for stores and data flows are expanded to find matching components for data flows to and from DFDs and PSpecs. Each definition may have different levels of expansion. Teamwork enables a user to specify the extent of DDE expansion to be checked. A user has a choice of a DDE being checked for an expansion of three levels or unlimited levels below the DDE name. If the DDE has more than three levels of expansion and the requested check is on three levels, then errors may be reported that would not have been had the check been on unlimited levels of expansion.

PSpec checking

The syntax of the data flows in and out of all PSpecs is checked when a user requests that the subtree of a DFD be checked. Teamwork also allows the user to request that only a particular PSpec be checked. PSpec checking reports if balancing of the PSpec and the corresponding primitive process is not maintained. PSpec checking does not include the body section of a PSpec so that a user can choose how to write the PSpec description. There is no checking that the inputs and outputs of the primitive process represented by the PSpec are exactly those included in the body of the PSpec.

Numbering and labelling

Teamwork also provides support for some numbering conventions used in Structured Analysis, especially those related to the numbering and naming of levels, processes, and process specifications. All the conventions described in Section 5.2.3 are supported by Teamwork SA/RT.

Limitations

In Teamwork (version 4.1) for Structured Analysis, the first time a child data flow diagram is created, it inherits the data flows from its parent process in that a list of the input and output data flows is produced at the child DFD. However, if changes are made to the parent process at a later stage, the tool fails to update the inherited data flows after the change. In other words, the inheritance process is not dynamic and automatic after the creation of a child data flow diagram. The feature of inheritance in Teamwork (version 4.1) is not useful since Teamwork (version 4.1) does not actually check that balancing is valid unless the check is requested explicitly.

5.3 Structured Analysis Enhanced With Real-Time Extensions

In the early 1960s, data processing systems were predominant and traditional Structured Analysis, as pioneered by DeMarco [10], was designed to meet the needs of such systems. With the development of smaller and significantly less expensive computers, whole new types of applications became more prevalent than data processing systems. Such new applications include reactive systems.

Reactive systems can be defined as systems having modes or states, inputs and outputs with processes being enabled and disabled at various times. They are systems that have no time constraints on them but rather whose order and value of outputs are determined by the order and sequence of inputs [14]. For systems with such properties to be encoded in SA, extensions were made to pure SA which provide for the specification of reactive systems.

The usage of "real-time" in SA/RT is a misnomer because the systems which are catered for by SA/RT are actually just reactive systems which may not have quantitative time constraints. However, the term "real-time" and the abbreviation "RT" will still be used in this chapter because the terminology is already well established and its meaning is well understood in its community of users. The flavour of SA/RT that will be described here is based on DeMarco [10] and Hatley/Pirbhai [15].

Structured Analysis with RT extensions is redefined to include State Transition Diagrams, State Event Matrices, and Process Activation Tables among its notational tools.

5.3.1 Notation Used in RT

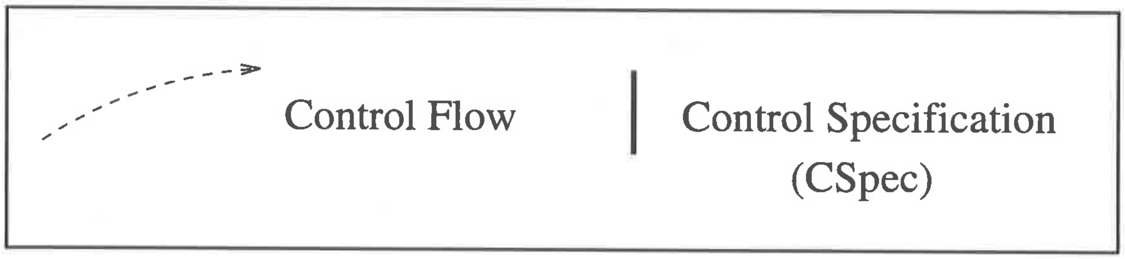


Figure 5.11: RT elements of a Data Flow Diagram

With the real-time extensions, Data Flow Diagrams have been supplemented with Control Flows and Control Specifications (CSpecs). Figure 5.11 shows the additions to the notation. Control flows are analogous to data flows except that they carry packets of discrete-valued data. Discrete signals are signals that can have one of a finite number of arbitrary known values [15]. Control flows are drawn as dashed directed arcs. They serve to guide the way in which processes do their work as well as to trigger a process to start working. Therefore, input control flows can be processed to generate either output control signals or process controls. Process controls are also called process activators because they can activate and deactivate processes. In the original SA, processes started transforming their input data flows when the latter were all available. In SA/RT, processes are not activated or deactivated by the control flows entering them because this is done by process activators within Control Specifications. Control flows must originate from a process or a terminator and must terminate in a process or in a CSpec. Control flows may be generated by processes, in which case, the values of control flows are set in the processes.

In SA/RT, the Control Specification is the mechanism that indicates how output control flows and process controls are generated from input flows. Thus, a CSpec is the control counterpart of a PSpec. However, CSpecs are usually associated with the DFD where they occur whereas a PSpec is associated with a primitive process. Other processes, whose activations are not controlled by process controls, are enabled each time there is sufficient data at their inputs to perform the specified function. A CSpec is represented by a vertical bar in a DFD (see Figure 5.11) and it is further fully detailed separately in any of several forms: a Process Activation Table (PAT), a State Transition Diagram (STD), a State Event Matrix (SEM), and a Decision Table (DT). In other words, a CSpec is further decomposed in one of these objects. The form of the CSpec object depends on the processing of the control flows needed. Each CSpec object must be associated with only one bar symbol. Control Specifications may appear at any level of decomposition. Control flows that are inputted to a CSpec are used by the CSpec and control flows that are outputted by a CSpec may be used as input to another CSpec or to a process.

A SA/RT DFD may also have control stores which are denoted just like data stores except that they contain control information. Control flows may also be drawn to and from control stores and they do not need to be named since they implicitly take on the name of the control store.

Therefore, a SA/RT Data Flow Diagram consists of processes that can be interconnected with data flows and control flows. Data flows join processes with other processes, terminators and stores. Control flows can connect processes, control stores, and terminators to other processes, control stores and CSpecs.

Data Dictionary

Definitions of control stores and control flows must be included in the Data Dictionary. A control flow may be made up of "control elements", therefore the latter have to be defined as well. A Data Dictionary entry for a control flow needs to indicate its attributes. The entry must indicate that it is a control flow and is discrete valued and whether it is a control element.

Control Specifications Objects

Control Specifications are used to specify the finite state machine behaviour of systems. A finite state machine can either be categorized as a combinational machine or a sequential machine. In the former, outputs are determined solely by current inputs; that is, there is no memory. Process Activation Tables and Decision Tables fall in that category of finite state machines. In SA/RT, the definition of a Decision Table is slightly augmented from its definition in original SA to cater for control flows as well. In SA/RT, a Decision Table (DT) describes the various combinations of input control signals and their results, i.e. the value of the output signals for each combination. A Process Activation Table (PAT) is a more sophisticated version of a DT. A PAT is a tabular description of the activation of processes based on certain events or combination of events and their order of activation (if order of activation is an issue). An event can be a control flow taking on a particular value or the truth value of a relational expression. A PAT cannot be used to describe activation of processes across the boundaries of one or more DFDs.

A sequential machine depicts control behaviour whereby outputs are determined by both current and past inputs. The concept of memory is represented in the form of states of a system. State Transition Diagrams (STDs) and State Event Matrices (SEMs) specify sequential machine behaviour. A STD represents the states of a system and the event or combinations of events that allow transition from state to state. Usually, when a system makes a transition from one state to another upon the occurrence of an event or combinations of events associated with that state, the system simultaneously takes corresponding action that can either be setting the value of a control flow or activating a process. Transitions may return to the state they left. The input control flows to a CSpec usually represent the events and the output control flows represent the actions. Therefore, STDs have four components: states denoted by rectangles, transitions denoted with directed arrows, events denoted by names on the arcs of transition, and actions denoted by names on the transition arcs. Labels on the transition arcs are of the form "*Event/Action*".

Outputs from CSpecs may also be fed into another CSpec with combinational machine behaviour. For instance, output control flows from a STD (in the form of actions) may be fed into a PAT.

An SEM is a tabular representation of a STD and may be used when the number of states and transitions are large.

5.3.2 Levelling and Balancing

Levelling

Levelling is maintained with the RT extensions in that the CSpec objects are the decomposition of the CSpec vertical bar. However, a CSpec object is not drawn on any of the child DFDs at which the CSpec occur. The CSpec object is specified separately and takes on the name and number of the CSpec it describes, just like a PSpec does for a primitive process. Control flows may be drawn at the Context Diagram from terminators. CSpecs cannot appear on the Context Diagram.

When a parent process is deactivated, it follows that the processes in the child DFD are also deactivated.

Balancing

Balancing also applies to the RT extensions. Whenever control flows are drawn to or from a process, they must be inherited by the child DFD. Control flows terminating in a CSpec and leaving the CSpec must be shown as inputs and outputs respectively in the associated CSpec object. Since control flows may be composed of control elements, their definitions in the DD must be expanded to enable balancing.

5.3.3 Conventions for SA/RT

The naming convention used with data flows also holds for control flows; that is to say, control flows are named with appropriate descriptive names. CSpecs are named with a noun – describing the kind of processing on the control flows – tagged to the acronyms STD, SEM, DT, or PAT according to the CSpec object they are meant to represent. Control stores are named with a noun describing their contents.

The name of a CSpec object is the number and the name of the CSpec bar which it decomposes. Its name must also indicate the level of the DFD at which the CSpec occurred. Figure 5.12 illustrates how Process Activation Tables are to be labelled. A similar set of labelling rules apply for State Transition Diagrams and Decision Tables.

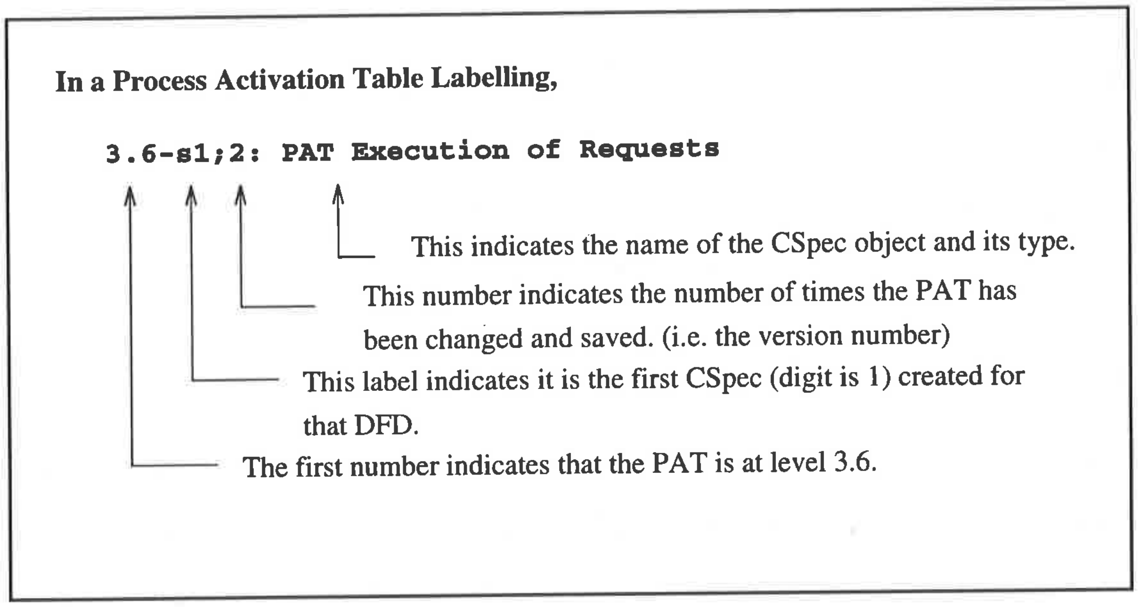

In a Process Activation Table Labelling,
3.6-s1;2: PAT Execution of Requests
This indicates the name of the CSpec object and its type.
This number indicates the number of times the PAT has been changed and saved. (i.e. the version number)
This label indicates it is the first CSpec (digit is 1) created for that DFD.
The first number indicates that the PAT is at level 3.6.

Figure 5.12: Process Activation Table labelling

5.3.4 Rules of Thumb for SA/RT

CSpecs should only be used if necessary and they should be used at the higher levels of DFDs. A CSpec should be used at a level where all its inputs are already present and most of its outputs are to be used.

The same principles as used for naming processes and flows should be used in naming states, events, and actions in a State Transition Diagram.

If a DFD has, say, six processes and four CSpecs, then some of the CSpecs can conceivably be eliminated or combined. The number of CSpecs per Data Flow Diagram must be kept to a minimum. Also, Decision Tables and Process Activation Tables should be minimised by combining similar rows and using "don't care" conditions for those rows whose entries have no effect on the consequences.

5.3.5 SA/RT development

Modelling a system in two steps might be wise for novices in the usage of the methodology: starting with the data flows first and then adding the control elements. This two step approach is encouraged because there is often confusion about whether a piece of information should be data or control.

5.3.6 Tool Support for SA/RT

Teamwork SA/RT provides support for the RT extensions to Structured Analysis. Teamwork SA/RT supports balancing of a DFD to its CSpecs by checking that the control flows connected to processes and CSpecs on a DFD match the input and output control flows in the CSpec objects. A user may opt to have only the specified DFD or its subtree checked as well. A subtree check of a DFD also validates the CSpec objects. The DDEs used in the CSpec objects must have definitions in the DD. Teamwork SA/RT detects CSpec object syntax and errors in states, events, and action expressions in STDs. This tool also checks that each control flow and its components are defined in the Data Dictionary.

Teamwork automatically provides support for the labelling of CSpec objects, as described in Section 5.3.3, and therefore, no explicit checks need be asked.

5.4 Worked Example

This section describes how the principles of SA/RT can be used to model parts of the SIS-Telereg system. First, the system context is established. The external agents that interact with the system are the registrar's office, cashier office, admissions office, academic departments, schools and faculties, and students. They are the net originators and receivers of data to and from the system respectively. They will be the terminators. The

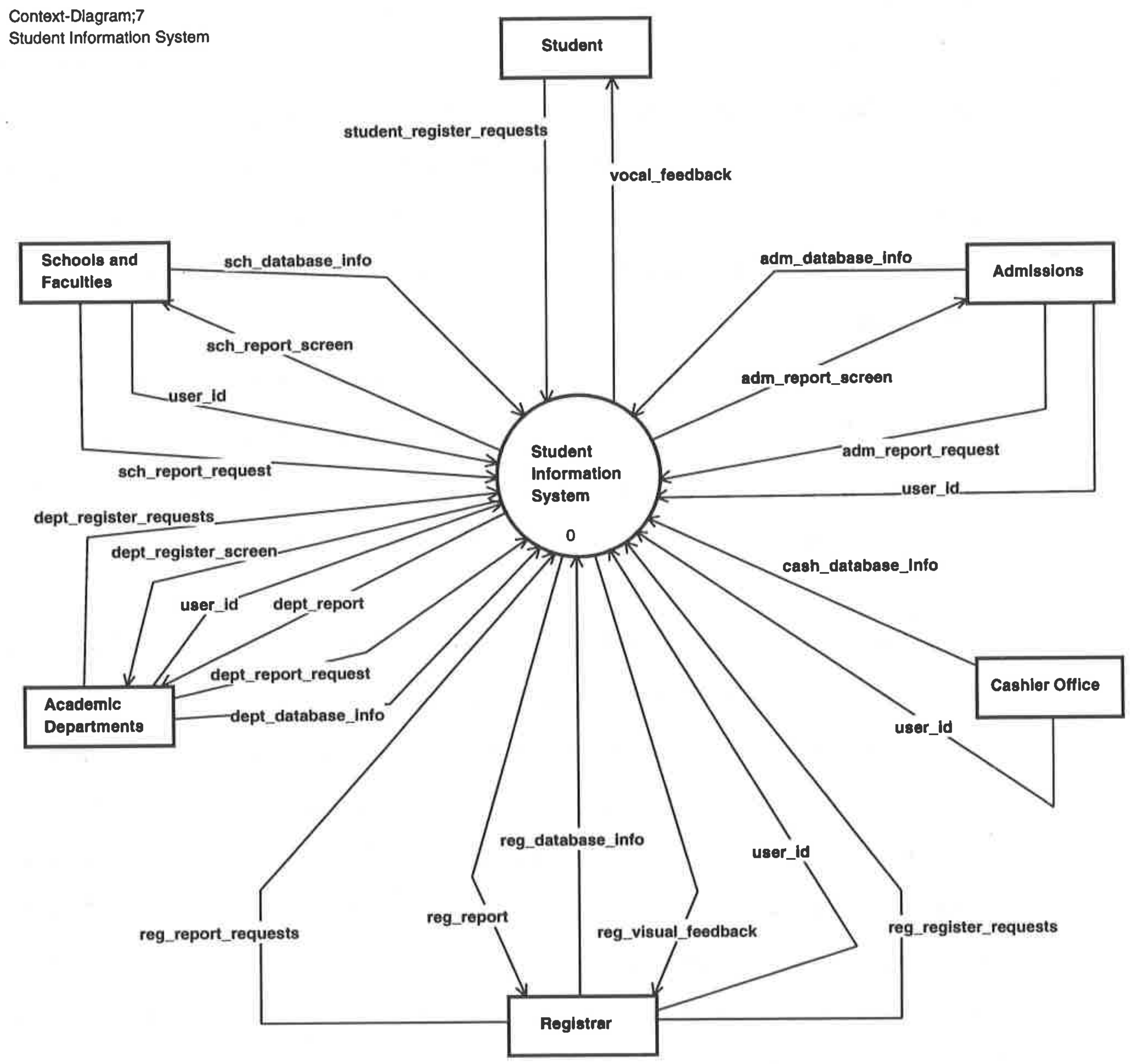


Figure 5.13: Context Diagram

three primary functions of the system are to update the database of students and faculty information, to allow for student registration of courses by either a student herself or the registrar's office, and generating of reports at the request of the various administrative offices. Therefore, to reflect these functions, the data flows are organized such that they can be database information, report requests, report screens or printouts, registration requests, and registration feedback. A user identification is also a necessary data flow because there are levels of security in the system that correspond to varying levels of access to the system. The Context Diagram, shown in Figure 5.13, depicts the system and its context.

While it is true that data such as a student identification will start the process of validating it, that piece of information is not a control flow. Otherwise, all data is control because their presence at a process allows it to do work. There is a tendency among new practitioners to make most data a control flow because it enables transformation at a process.

Data flows "vocal_feedback" and "reg_visual_feedback" are defined as aliases to "registration_feedback" because they have the same components even though they originate from two different sources. More importantly, they reflect the difference in the format of the data from two different sources. Figure 5.14 describes these data dictionary entries.

At the first level of decomposition (see Figure 5.15), the system can be partitioned into four processes, three of which correspond to the three main functions of the SIS-Telereg system and the fourth process determining the security access level for a given user of the system. To satisfy the purpose of the document, which is primarily a model of the registration process within the SIS system, processes 1 and 2 will be expanded to one more level of decomposition since they do not relate to the focus of this specification exercise. Process 4 describes how the security access level is determined and since it is not further decomposed into a child DFD, its processing is described in a PSpec. Process

```
reg_visual_feedback (data flow, alias) =
    registration_feedback
    * These are the visual version of the messages that are given out by
    the system. These messages are exactly identical to the ones issued
    by Telereg telephone registration system except that they are
    integrally displayed on a computer screen. *

registration_feedback (data flow) =
    [ request_id_msg
      | invalid_id_msg
      | bye_msg
      | last_name_msg
      | ...
      | register_msgs
    ]
    * These are the kinds of verbal messages that the SIS-Telereg
    system can give out to the students. *

vocal_feedback (data flow, alias) =
    registration_feedback
    * These messages are given by the system verbally through a phone
    receiver in response to the registration requests. *
```

Figure 5.14: Aliases in Data Dictionary entries

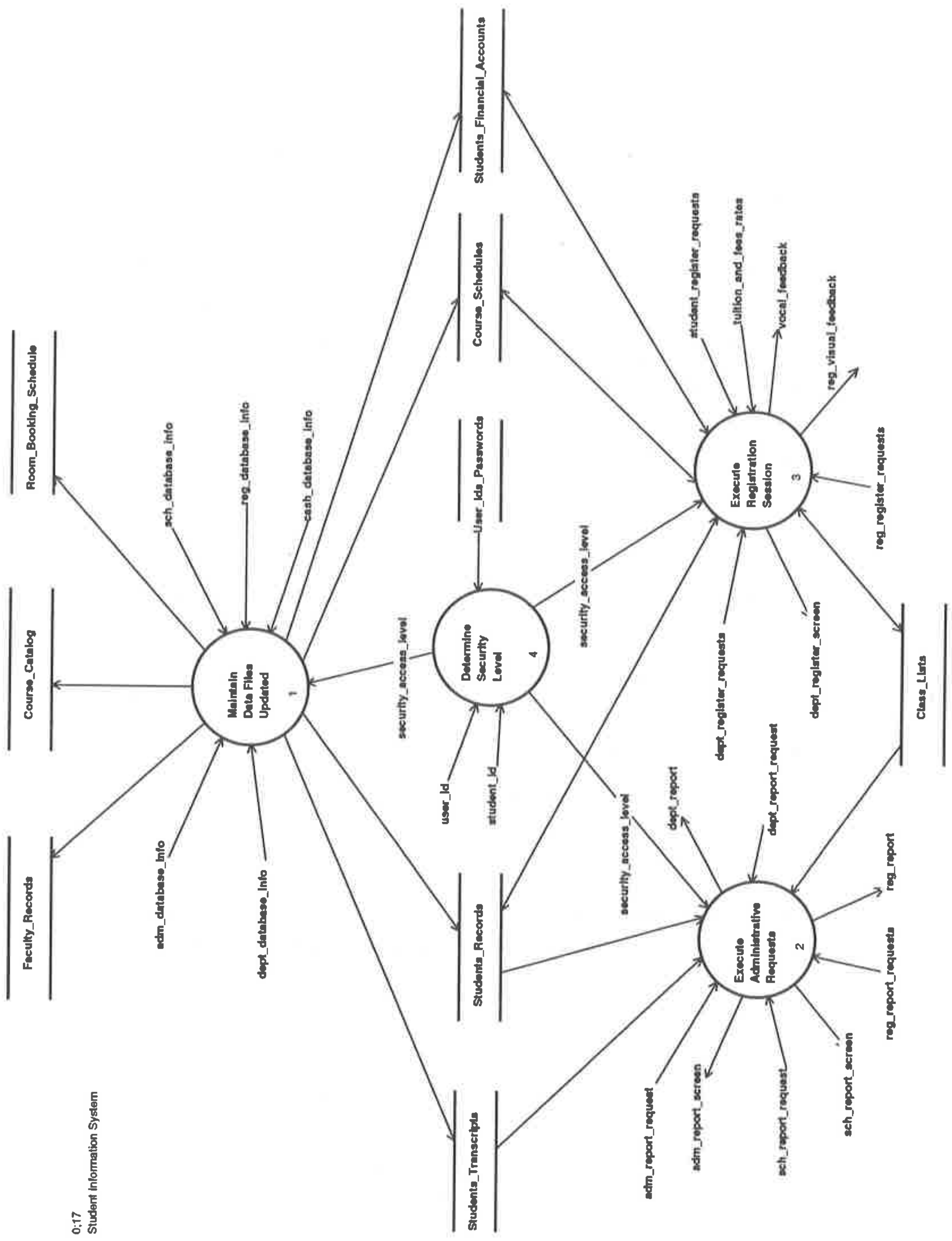


Figure 5.15: Level 0 - Student Information System

3, which deals with the registration process, will have three more levels of decomposition.

Process 1 takes database information from various sources and updates some data files. This might raise the question of whether this process is needed since updating a data store hardly qualifies as processing and the data flows could be drawn as going into the data stores directly. The question arises because the SIS-Telereg system does not perform much processing except for computation of fees or grades for example. However, given that the system is by nature a central database, its main function consists of updating data files and, as such, justifies the use of processes to describe additions, deletions, and updates to the database.

Also, at the first level, data stores which have been updated by process 1 are used by other processes (2 and 3) for generation of reports and execution of registration requests. This is shown on the DFD.

Since the model was created in Teamwork SA/RT, the numbering and labelling of the child DFD of the system bubble and the processes of the child diagram are automatically provided. Balancing, however, is maintained by checking the input and output data flows on the context diagram with those on the child DFD and using the decomposition of the DDEs for these flows.

Figure 5.16 is the child data flow diagram of process 3 in the level 0 data flow diagram. The processes on this data flow diagram are not connected with one another because no output from any one of these bubbles is used as input to the other bubbles. However, there is a certain order in which the processes can be activated and this is described in the two CSpecs. In this DFD, looking up entries in the DD is necessary to ensure balancing because many of the input and output data flows are components of the data flows from the parent process.

The state transition diagram in Figure 5.18 depicts the three states in the registration process and how a user moves from one state to another. The diagram is to be read

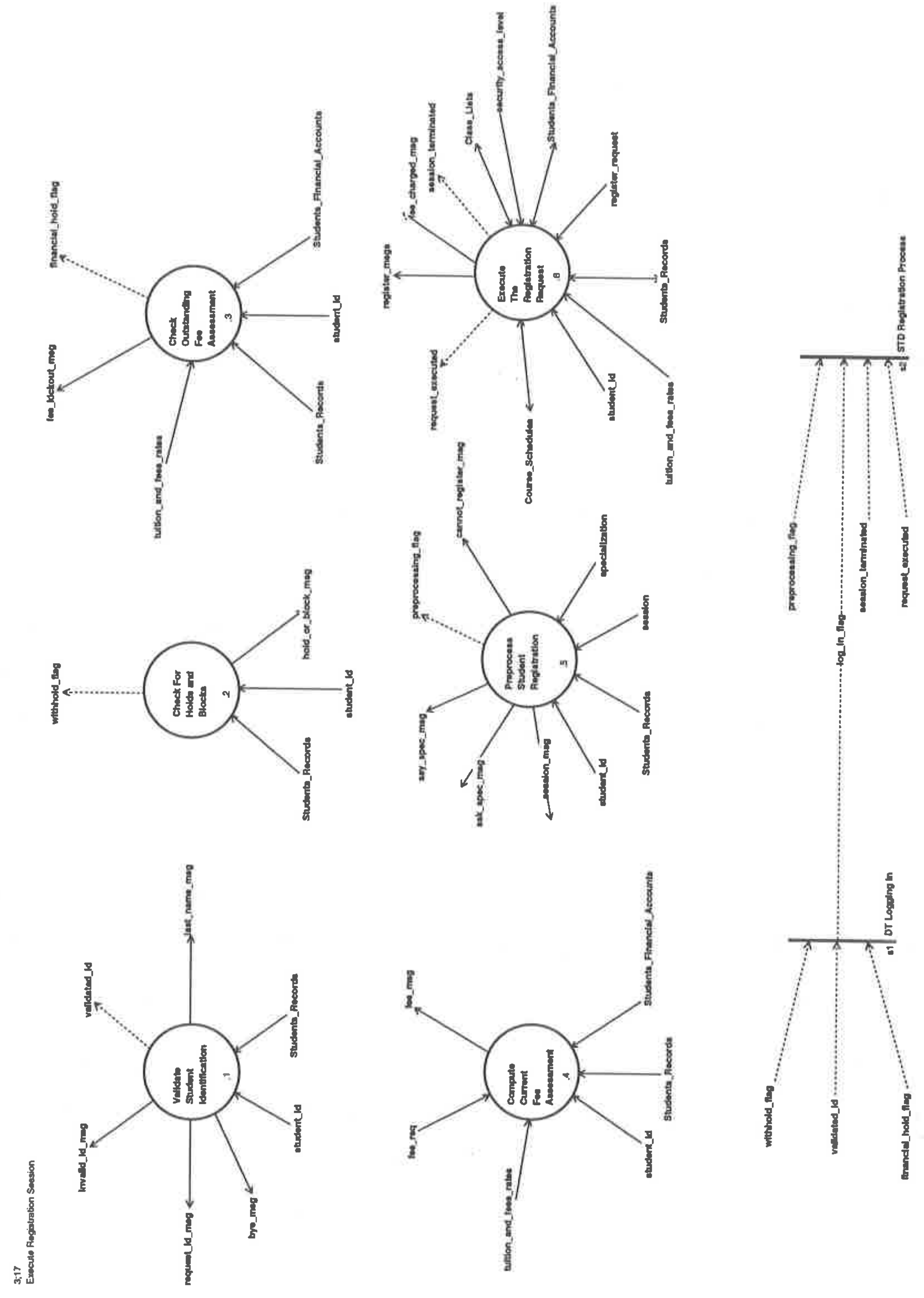


Figure 5.16: Level 3 - Execute Registration Session

3-s1;5
DT Logging In

financial_hold_flag	withhold_flag	validated_id	log_in_flag
"FALSE"	"FALSE"	"TRUE"	"TRUE"
			"FALSE"

Figure 5.17: DT Logging In

in conjunction with the data flow diagram in Figure 5.16. The registration process starts with the user providing her student identification. This piece of information is validated and the student's record is checked for financial holds and disciplinary blocks as well as for an outstanding fee balance. These tasks are performed by the first three processes. The decision table in Figure 5.17 shows the resulting outcome of the outputs from these processes. The blank entries in the DT stand for "don't care" conditions. If the final outcome is positive, then the student moves from the "Logging In" state to the "Preprocessing" state where the student's eligibility to register and access date are checked upon receiving the session for which registration is sought (process .5). At this state, the student can request her fee balance (process .4). If the student successfully passes the current state, then she can move to the "Processing" state where she can request multiple registration transactions (process .6). This state is exited when the registration process is over. Process .6 from the level 3 data flow diagram is further decomposed but it is not discussed here.

Since using Structured English made the PSpecs look like pseudocode, appropriate words and phrases were added to make them read more easily. The PSpec for primitive

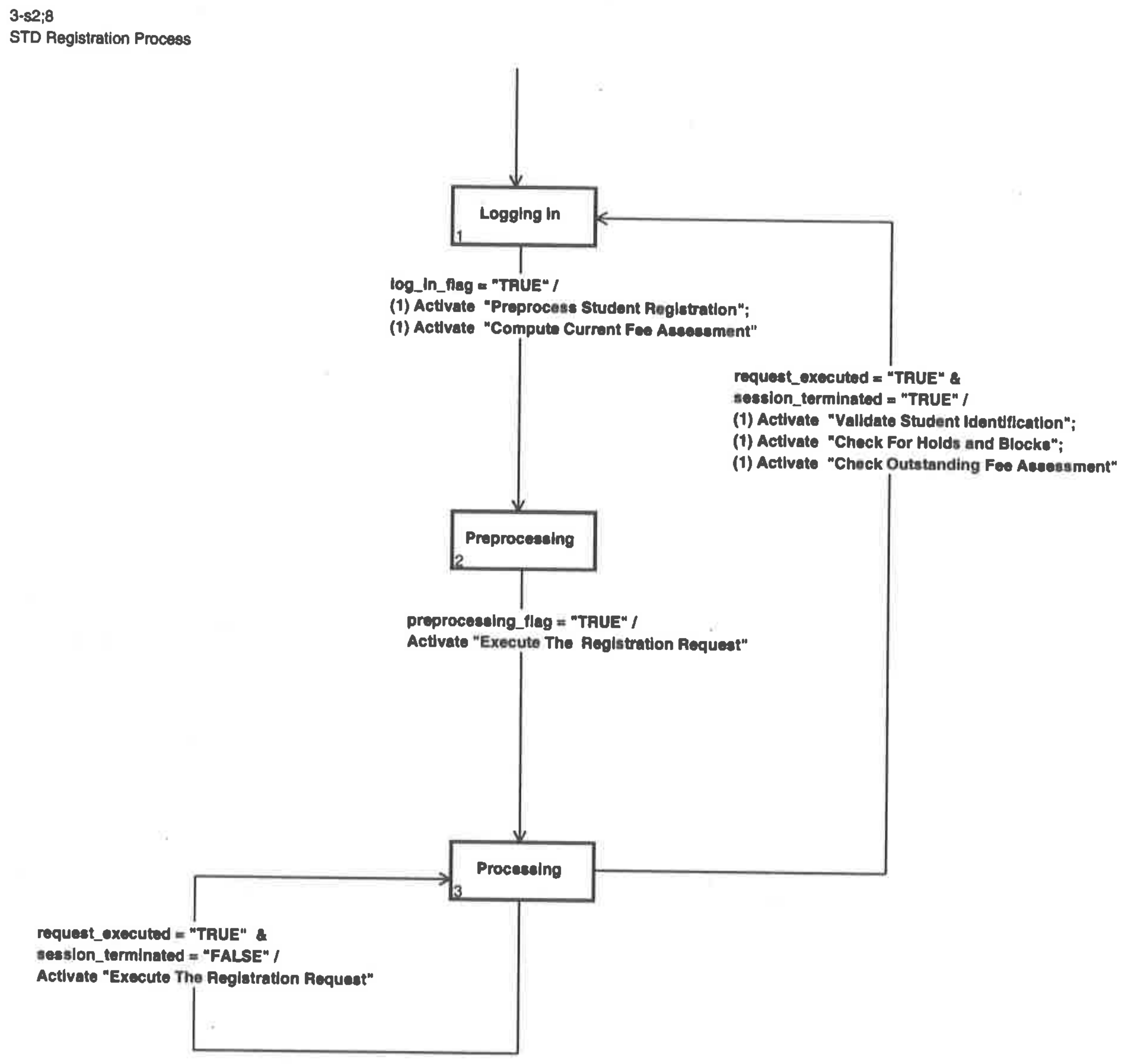

3-s2;8
STD Registration Process
Logging In
1
log_in_flag = "TRUE" /
(1) Activate "Preprocess Student Registration";
(1) Activate "Compute Current Fee Assessment"
Preprocessing
2
preprocessing_flag = "TRUE" /
Activate "Execute The Registration Request"
Processing
3
request_executed = "TRUE" &
session_terminated = "TRUE" /
(1) Activate "Validate Student Identification";
(1) Activate "Check For Holds and Blocks";
(1) Activate "Check Outstanding Fee Assessment"
request_executed = "TRUE" &
session_terminated = "FALSE" /
Activate "Execute The Registration Request"

Figure 5.18: STD Registration Process

process 3.1 is described in Figure 5.19.

It may happen that a functionality is described at one level of a DFD and the same functionality with same inputs and outputs is needed at a different level. While these processes are distinct processes, they are equivalent in functionality. Instead of duplicating the PSpec processing description, the example in Figure 5.20 illustrates a convention that can be used in cases where PSpecs are equivalent.

5.5 Difficulties in Learning SA/RT

The principles of Structured Analysis are relatively easy and simple to understand. However, learning to assimilate all of them to apply to a full-fledged model may not be an easy task for a new practitioner of SA. Indeed, the lack of guidelines on how to start and develop a model can be a setback for a new user since the latter may not know how to apply the principles. Although textbooks vary in what they claim is the key to starting a model, it might be easier to start with the context diagram, so that all the net input and output data flows and control flows are established, and to partition from then on. Often, a model must be started from scratch. A correct model is built incrementally; each time that a model is changed as an aspect of SA is mastered, other aspects have yet to be learnt. With each model, the gap between reading about Structured Analysis and actually using it well gets smaller. Even if the first few attempts at modelling a system are wrong, they are not wasted because they help the specifier acquire a better understanding of the system; ideas take time to evolve.

When a new user draws data flow diagrams for the first time, it is easy to confuse data flows and control flows. Often, it is wrongly perceived that since a process can get started when it receives a piece of data, then that piece of information should be a control flow. If the data is not discrete-valued, then it cannot be a control flow. Even

```
P-Spec 3.1;8: Validate Student Identification

NAME:
3.1;8

TITLE:
Validate Student Identification

INPUT/OUTPUT:
bye_msg : data_out
validated_id : control_out
invalid_id_msg : data_out
request_id_msg : data_out
last_name_msg : data_out
student_id : data_in
Students_Records : data_in

BODY:
attempt + 1
validated_id = "FALSE"
IF (student_id format = INVALID AND attempt < 3)
    give request_id_msg
ELSE IF (attempt >= 3)
    give bye_msg
ELSE IF (attempt < 3)
    search student_id against Students_Records
    IF (student_number OR birth_date = INVALID)
       give invalid_id_msg
       give request_id_msg
    ELSE
       IF (found)
          give last_name_msg from student_id.Students_Records
          validated_id = "TRUE"
       ELSE
          give invalid_id_msg
          give request_id_msg
```

Figure 5.19: P-Spec 3.1;8 Validate Student Identification

```
P-Spec 3.4;7: Compute Current Fee Assessment

NAME:
3.4;7

TITLE:
Compute Current Fee Assessment

INPUT/OUTPUT:
fee_msg : data_out
Students_Records : data_in
Students_Financial_Accounts : data_in
student_id : data_in
fee_req : data_in
tuition_and_fees_rates : data_in

BODY:
EQUIVALENT TO 3.6.4.9 except for data_in request_executed and
   session_terminated.
```

Figure 5.20: Equivalent Process Specifications

if the data is discrete-valued, it may or may not be a control flow. To avoid confusion between data flows and control flows, DFDs must first be drawn with only the data flows. Then, control flows can be added. This two step approach prevents the tendency to make just about every piece of data a control flow.

Understanding the proper usage of control flows and their true roles can be difficult and, therefore, be a major stumbling block in modelling a system. It is quite straightforward to understand that control flows which are drawn into CSpecs can be used in Process Activation Tables, Decision Tables, and State Transition Tables. But it is not clear what control flows drawn flowing into processes would be used for; it means that the information carried by the control flows is used by the process which receives it as input. For instance, depending on whether the value of a control flow named "validation_flag" is "true" or "false", a process receiving this control flow as an input may output different messages. The process would test the values of "validation_flag" in its PSpec.

Some new users have no reservations using control flows but they often fail to use CSpecs to show how control flows are processed. Not using CSpecs to describe how control flows are processed is like not using PSpecs to describe how data is transformed in processes. New users tend to avoid using an STD or SEM; they settle to using a PAT because the latter is easier to understand. While it is true that part of what is captured in an STD or SEM can also be captured by a PAT, namely the activation of processes and their order of activation, PATs, nevertheless, fail to capture the requirement that a particular event may only occur in a particular system state and that an event may cause transition from one state to another. If sequentiality is an integral part of the requirements, then it must be clearly specified. A PAT is not adequate to show sequentiality. Other users try to depict the sequentiality aspect of a model in the data flow diagrams by drawing data flows from process to process. Consequently, these superfluous data flows cannot be named properly because they were not needed in the first place.

The DFDs then look like flow diagrams. Also, new users may be under the misguided impression that processes cannot be separate entities and somehow have to be connected to each other. The examples in DeMarco [10] can be quite misleading because most of them show processes which are always connected and which actually pass on valid output data to other processes. However, it must be borne in mind that DeMarco's approach is best suited to data processing systems.

Enough details must be included in the context diagram so that it matches the system realistically. The context diagram should clearly describe the functionality of the system. Hence, the way data flows are grouped together is important. Data of the same type are grouped together so that their descriptive names give an indication of the functions of the system.

A data flow diagram is distinct from a flow chart. A data flow diagram focuses on data; it shows where data comes from, how it is changed, and how it is used. On the other hand, a flow chart depicts how a processor acts on data. In other words, a DFD is a depiction of the functionality of the system from the point of view of data whereas a flow chart is a design documentation from the point of view of the system [10].

5.6 Strengths and Weaknesses of Structured Analysis

This section presents an evaluation of Structured Analysis.

5.6.1 Strengths

1. **SA has less potential for ambiguity than a natural language specification approach.**

   Structured Analysis offers the advantage of writing requirements in a semi-formal manner with graphical notations that have precise syntax and loosely defined semantics. Since natural language is confined to the process specifications and data dictionary entries, there is less potential for ambiguity than if the specification is completely written in natural language. Since this approach is relatively mature compared to other specification techniques, e.g. formal techniques, there is already an established set of conventions and rules of thumb which serve to guide in specifying readable and adequate models. Also, these informal guidelines aid in a more systematic development of the requirements than if they were specified *ad hoc* in natural language.

2. **SA can enhance human readability and comprehension.**
   The use of graphics enhances human readability and understandability whereby improving communication with the user. Hence, it facilitates the review processes to verify that user requirements have been correctly interpreted. Good communication can also potentially help discover redundancies and inconsistencies in the specification document. Hence, Structured Analysis offers some means to ensure that requirements are self-consistent and complete, thus ensuring that the statement of requirements is a sound basis to start design.

3. **SA encourages a rigorous understanding of the requirements.**
   Structured Analysis causes codification to be moved forward into the analysis phase. This helps bridge the gap between requirements analysis and specification stage and the design and implementation stages. Codification also compels the specifier to rigorously investigate the application domain in order to understand the requirements. Consequently, it is likely that the requirements will be better understood before the design process is started.

4. **SA has tool support.**

   Since the notational tools in Structured Analysis have a well-defined syntax, tools can and have been built to check the syntax of an SA model. Even though the semantics associated with the notation is not well-defined, they enable some type of checking. While the internal consistency of an SA model cannot be verified with tool support, balancing of data flow diagrams can be fully supported by some CASE tools. Some CASE tools such as Teamwork also support data consistency in the data dictionary. Often, levelling facilities are offered as part of the modelling tool. Actually, the lack of precise semantics is quite advantageous in that the specifier need not worry about the fine details about what a notation means before it can be used.

5. **SA offers different levels of abstraction of the system.**

   Through levelling, Structured Analysis offers a top down analysis of the system. Therefore an SA model can be read at different levels of details. "The data-flow model is advantageous in showing the synchronization of functions with the data paths that flow between them, as well as an excellent mechanism for progressive system decomposition from the most to the least abstract levels of detail." [12].

6. **In SA, identification of inputs and outputs in the process specifications facilitates testing.**

   While an SA model is not testable, its process specifications are. Process specifications are in natural language and are structured in such a way that they can potentially aid in testing. Process specifications separately identify inputs and outputs to a process before describing the processing done on inputs to generate outputs. The inputs and outputs sections can be useful for testing.

7. **With its real-time extensions, SA/RT is especially suited for systems with finite state machine behaviour.**

   Structured Analysis with RT extensions is well suited for describing finite state machine behaviour. Structured Analysis adequately captures two out of three possible ways of modelling a system; namely, the information processing or functional model and the control or state model. Structured Analysis does not model the information view of a system. Data flow diagrams describe functional requirements while CSpec objects depict flow of control.

5.6.2 Weaknesses

1. **SA often imposes design structure.**

   One of the major drawbacks with Structured Analysis is that an SA model may not be decoupled from design. Structured Analysis not only causes an analyst to specify requirements but to partition what is being specified as well, thereby increasing the risk of stepping into design issues. Therefore, the specification document is not as maintainable as it would have been had it been decoupled from design.

2. **The lack of precise semantics in SA may cause ambiguities.**

   The lack of precise semantics in the methodology means that there might be ambiguities in the specification model since it is not clear what things mean precisely. Also, it makes it difficult to build tools to do any kind of useful analysis on the model beyond balancing data definitions.

3. **SA models are not easily maintainable and modifiable.**

   Traditional Structured Analysis focuses on the functional and organizational decomposition of data. By partitioning the requirements to offer a hierarchical view of the system, the methodology indirectly introduces design structure in the model.

If some functionality is to be removed from an existing SA model, it is likely that the functionality has to be removed from various places and levels in the document to reflect the change. Moreover, tools, such as change bars and strikeouts, which identify changes made to the model are not available. Hence, an SA model is not readily maintainable. Therefore, SA cannot also promote code re-use because the functional requirements of systems are bound to change with time as the systems are expanded.

4. **SA models are not testable.**

   SA models are not testable; what is testable is the natural language for the process specifications. However, the model is the bulk of the requirements. Some practitioners [5] say that models are usually put in the appendix sections of specification documents and they are not used to write unambiguous testable requirements. For the most part, test cases are generated from the natural language specifications. However, if the data flows in the model do not correspond to the real data entities in the system, the inputs and outputs of the process specifications are not useful for testing.

5. **Structured Analysis may not be suitable for large systems.**

   Using Structured Analysis implies that a certain amount of design is derived as part of requirements analysis. While this might be acceptable for small software projects, it becomes increasingly undesirable as the size of systems grow. Large systems need specifications that are well removed from design. Also, it is more difficult to determine when to stop partitioning large systems than it is for small systems. Further, there is a lack of tool support for large systems. Thus, Structured Analysis is not a good method for large systems.

6. **SA does not adequately address concurrency.**

   The DeMarco and Hatley/Pirbhai approach to SA/RT does not address concurrency. This is a drawback because systems such as the SIS-Telereg system, banking systems and so on are prevalent concurrent systems. However, the Ward/Mellor [33] approach to SA recognises that concurrency needs to be modelled and offers a notation for it and and gives heuristics for choosing an optimum number of tasks. To allow concurrency in the DeMarco and Hatley/Pirbhai approach to SA/RT, the concept "spawn" can be introduced and be defined as "creating an exact copy of a process(es) or DFDs but with the data flows containing the unique values given by a particular user".

5.7 Conclusion

This section describes some remarks on Structured Analysis and evaluates SA/RT as a semi-formal approach.

5.7.1 Remarks About Structured Analysis

Structured Analysis forces a rigorous study of the user area which can provide a first cut at understanding the system. Since the notations in SA are mostly graphical, with some kind of semantics attached to them, pictures of the system can easily be drawn and cast aside if they are wrong without this trial and error process being a very time consuming effort. If natural language were used to write the specification as the requirements are slowly understood, major rewrites of the specification may be needed. After the requirements are understood using SA, other techniques can be used to establish the specification document.

5.7.2 Evaluation of SA/RT as a Semi-Formal Specification Technique

Structured Analysis is a good example of a semi-formal approach to specification. It offers a good compromise between formal and informal specification methods. It is mostly graphical except for the Process Specifications and Data Dictionary entries that are in Structured English and English respectively. The graphical notations have a precise syntax and a loosely defined semantics. There are also many conventions and rules of thumb for SA that are already well established in its community of users.

The combination of Structured Analysis with other specification notations has become popular in recent years. For instance, Potter *et al.* [24] suggest using a combination of Structured Analysis and a formal specification language called Z. They suggest a combination where "data flow diagramming is used to structure a specification, and Z is used to capture the required behaviour of the components" [24]. Using Z as a formal notation has a lot of appeal because it will help reduce ambiguity where it is most likely to be found: in the process specifications. Potter *et al.* also suggest an alternative approach whereby Z is used to capture the requirements precisely first before they are modelled in data flow diagrams. In this case, they suggest that the approach is beneficial in that not much time is spent in producing a hierarchically correct model which might change as requirements become clearer.

Chapter 6

Structured Informal Specification Technique

6.1 Introduction

A more sophisticated kind of informal specification technique used in industry has been developed to capture requirements in a less *ad hoc* manner while still using natural language as the medium to communicate requirements. This kind of structured informal technique has great possibilities of becoming widely adopted because it is not as notationally constrained as Structured Analysis or formal specification language Z but it offers structure and there are obvious advantages from using it. The threads-based approach is the structured informal specification technique which will be described in this chapter. The threads-based approach is elaborated from ideas derived from Deutsch *et al.* [12] and my research association with Hughes Aircraft of Canada Ltd, Systems Division (HCSD).

The threads-based approach is based upon the concept of a *thread* which is defined as "a path through a system that connects an external event or stimulus to an output event or response" [12]. The threads-based technique therefore focuses on external stimuli and responses. A threads-based specification consists of a series of statements, each relating an external stimulus to an external response. The requirements are written to specify the required external behaviour of the system to be implemented. The threads-based technique is basically a methodology for writing requirements in natural language in a structured manner whereby the stimuli and responses are made explicit.

Section 6.2 defines the threads-based technique and describes the notational tools involved in this approach to software specification and the various conventions used to write requirements using this technique. Section 6.3 illustrates how the notation and conventions of the threads-based approach may be used to specify parts of the SIS-Telereg system. Section 6.4 summarises an evaluation of the technique. Section 6.5 concludes with some remarks about the technique and evaluates the threads-based approach as an informal specification technique.

6.2 Threads-Based Technique

The threads-based specification technique applies only to the functional requirements section of a system requirements document. A threads-based specification consists of a series of statements, each relating an external stimulus to an external response. The threads-based approach is a methodology that partitions the functional requirements of a system into basic specification units called **threads, capabilities,** or **conditions** and describes components of the system state in a **data dictionary**. The threads-based approach provides a functional view of the system where a stimulus results in a response or an update in the system state. The threads-based approach is limited to a single level of abstraction.

A set of documented conventions govern the functional specification which, written in natural language, uses no special mathematical or graphical notation. More importantly, the threads-based approach makes the stimuli to and responses from the system explicit and mandates the use of "shall" statements for any testable requirements. The requirements are written to give a black box view of the system. The requirements only have to be written and partitioned according to certain rules. The specification units and the data dictionary are the notational tools of this technique. Certain properties intrinsic to the technique are associated with the notational tools; they must be satisfied for the methodology to hold. However, it must be borne in mind that there is a fine line between what is to be an intrinsic property and a convention since this technique is a refinement of the *ad hoc* approach (see chapter 4).

6.2.1 Notation Used in Threads-Based Technique

Data Dictionary

A *Data Dictionary* (DD) is an essential tool in the threads-based style of specification. The data dictionary consists of an alphabetical list of terms that describe the entities that constitute the system state as well as the inputs to and outputs from the system. The data dictionary is an essential part of the specification and it is usually included at the end of the specification document. The data dictionary used in the threads-based technique is the same as the one described in Chapter 5 for Structured Analysis. Data dictionary entries are global to the entire specification.

Data dictionary entries may be made of components. Data dictionary entries are repeatedly partitioned top down into their components until the components are basic entities of the system or part of the terminology used in the problem domain. Definitions must be consistent with their application usage. Each data dictionary entry must be unique.

Data dictionary entries or items are referenced by enclosing them in angle brackets. In other words, all data dictionary entries are referenced as "⟨**data dictionary item**⟩". The equality sign (=) associates a DD entry with its definition. Comments may be used in the DD to clarify the entry.

Referencing Local Names

A similar set of requirements may apply to a set of stimuli instead of a single stimulus; consequently it would be convenient to refer to a set of stimuli by a single name. The same is true for responses. Local names are used to refer to a set of messages which are similarly processed. Local names are enclosed in square brackets in the format "**[local name]**".

Threads

Threads specify the actions performed by the system as a result of one or more stimuli. The notion of a "thread" as a specification unit is distinct from the conceptual notion mentioned earlier, i.e. "a path through a system that connects an external event or stimulus to an output event or response". In most cases, the word "thread" is used to signify the specification unit; if it is meant to stand for its definition as a path, this will be explicitly stated. In fact, thread specification units consist of paragraphs of requirements that are organised around the concept of thread paths connecting stimuli and responses. A thread has three mandatory sections: **STIMULUS**, **RESPONSE**, and **REQUIREMENTS**.

The **STIMULUS** section can only occur in a thread, and describes the stimuli for a thread. The stimuli, grouped into classes based on their functionality, are given in the form of their data dictionary entries. Each class of stimulus is associated with a local name which will be referenced in the **REQUIREMENTS** section. Local names do not have any significance outside the context of the thread in which they occur, just like the scope of local variables in a programming language is restricted to the function or procedure in which they occur. The sources that send the inputs may also be identified since multiple sources could have sent the input. If a stimulus originates from two different sources, it is listed as two data entries under the same local name provided the stimulus yields the same functionality irrespective of its source. Conversely, if two different stimuli may be sent from a single source, they are listed separately under different local names if they are processed differently.

Each class of stimulus or input is introduced with the sentence specified in Figure 6.21. *Any one* data entry associated with a local name is necessary for the local name

(System Name) shall satisfy the requirements of this thread upon the receipt of a [local name] of the form:

a. (Source) <Data Dictionary Item>
b. (Source) <Data Dictionary Item>
c. ...

where (System Name) is replaced by the name of the system and (Source) is optional to specify the source of the stimulus

Figure 6.21: Usage of local names in the STIMULUS section

to be referenced in the REQUIREMENTS section. There is no significance in the order in which data entries are listed under a local name. The sentence in Figure 6.21 may also be used if a class of stimulus consists of only one stimulus; that is, a local name is associated with only one data dictionary item. The composition of the data dictionary entries is specified in the DD, not in the STIMULUS section.

The STIMULUS section may also list internal events as inputs. Internal events include the "commit" of a data item, a timed event, or an event described by another capability.

The RESPONSE section specifies the possible responses that are given by the system. The destinations of the responses may also be specified. Responses are specified by their data dictionary names and they are grouped into classes. If a response is to be sent to two different destinations, it is listed as two different data entries. Also, if two different responses are to be directed to a single destination, they are listed separately. A local name is provided for each class of response. As with local names in the STIMULUS section, local names in the RESPONSE section are meaningful only in the context of the thread in which they occur. A local name is used to refer to *any one* of the responses that fall under the class designated by that local name. The order in which data dictionary entries are listed is arbitrary. Each class of response is indicated by the sentence specified in Figure 6.22.

(System Name) shall return a [local name] as follows:

a. (Destination) <Data Dictionary Item>
b. (Destination) <Data Dictionary Item>
c. ...

where (System Name) is replaced by the name of the system and (Destination) is optional to specify the destination of the response

Figure 6.22: Usage of local names in the RESPONSE section

The REQUIREMENTS section details all the requirements that relate to the function performed in the thread. Syntactic checks on the inputs are not described. However, any semantic checks to validate the values of inputs are described. This section also specifies the processing done when inputs are accepted and rejected. There is no implied order in which the requirements are written or executed. The requirements specify the processing done on each class of stimulus to generate the corresponding class of response.

Every requirement is written either as a "shall" statement or as a "will" statement.

A requirement is written as a "shall" statement if it is a testable requirement. Therefore, each testable requirement is indicated by a "shall". A requirement may be testable by either inspection, demonstration, or analysis. On the other hand, a "will" statement describes requirements that need not be met to satisfy contractual agreement. A "will" statement may describe a requirement which is not testable or may hint at an implementation preference. A "will" statement is also used to provide explanations or clarifications.

"Commit"s are used to signify that the change in system state is permanent as opposed to "update"s which describe temporary change in system state. Hence, "commit"s are done on "safe data" to ensure that their values are maintained if the system fails. "Commit"s of data entities are testable and are used in "shall" statements. Conversely, "update"s are used in "will" statements because they are not testable. The verb "retain" is used where the data may be lost after a system crash. The notion of retaining data is testable and is used in conjunction with a "shall". The verb "return" is used in threads whenever a response goes back to the source of the stimulus or set of stimuli triggering the response. If the destination of a response is different from the source of the stimulus or set of stimuli that triggered the response, the verb "send" is used.

System state and conditions names are enclosed in single quotes. For example, one would say '*the telephone connection is to be terminated*'. If the requirements refer to an entity in the system state whose value has been specified by some component of the data entries grouped under a local name in the STIMULUS or RESPONSE sections or if the requirements refer to an entity that has been obtained or derived from the stimuli, this must be indicated with the phrase "**...included in [local name]...**". For instance, one would say: *'There are <openings> in the new <section> included in [input change section]'*. The requirement specifies that "*[input change section]*" contained information on the entity "*<section>*".

Threads may not reference other threads. They may, however, reference one or more conditions and capabilities. Threads may not be referenced by capabilities or conditions. Whenever a specification unit refers to another, the phrase "**...as specified in \The Name of the Functional Block\...**" is used. For instance, if a thread makes reference to a condition which has been previously defined, the phrase "**...as specified in \The name of the Condition\...**" is attached to the description of the condition in the REQUIREMENTS section of the thread. For example, *'The student can access the registration process for the <session> included in [input session]' as specified in \Check Access Dates Condition* is a condition in a thread that refers to the previously defined condition "Check Access Dates Condition".

Conditions

Conditions express the system preconditions for a particular stimulus to trigger a particular response. Conditions have one mandatory section: a REQUIREMENTS section. The REQUIREMENTS section describes the factor or combination of factors that make the condition true. In other words, this section is written in terms of predicates. Conditions may be referenced by both threads and capabilities. Conditions may not reference any specification unit except for conditions.

Capabilities

A capability can be used as a packaging mechanism which groups conceptually related threads. A capability may also serve as a mechanism for specifying some processing that is common to several threads. For instance, a capability may be used to express all the requirements pertaining to the validation of a registration request in a registration system. Threads processing the various types of registration requests will then make references to the capability rather than repeating the same validation requirements in each thread. Capabilities consist of an OUTPUT section and a REQUIREMENTS section. There is no STIMULUS section because capabilities are referenced by threads or other capabilities.

The REQUIREMENTS section has the same function as it does in threads and therefore follows the same rules. The OUTPUT section corresponds to the RESPONSE section in threads except that, in threads, the destinations of responses are usually the sources of their stimuli whereas, in capabilities, since there is no stimulus, the destinations of outputs are usually different from the sources specified in the threads which reference the capabilities. The verb "send" is used with the outputs. The OUTPUT section is optional. Local names used in the OUTPUT section must match the local names used in the RESPONSE section of the threads that refer to the capability.

Capabilities may reference other capabilities and conditions. They may not reference threads but threads may reference them.

Consistency checks

As threads and capabilities are able to refer to one another, their local names must be consistent. Data dictionary entries must also be consistent throughout the DD and with their usage in the specification document.

6.2.2 Conventions for Threads-Based Technique

Functional Block Titles

Each specification unit is given a title that distinguishes it from other specification units. Specification units describe some kind of processing on the system objects; therefore they are named with a verb and a noun prepended to the suffix "**Thread**", "**Capability**" or "**Condition**". The verb is meant to describe the action done to the object specified by the noun.

Hence, thread titles are of the format '**Verb Noun Thread**'. Condition and capability titles are of the forms '**Verb Noun Condition**' and '**Verb Noun Capability**' respectively.

Data Dictionary Entries

Data dictionary names may include blanks, hyphens and underscores. Data dictionary entries for outputs and responses are named with the prefix "**msg**" if they are messages. The plural of a data dictionary entry is indicated by adding an "**s**" at the end of the entry in angle brackets; for example, <student identification>s. Comments may be enclosed between asterisks.

Local Names

In the STIMULUS section, local names that are enclosed in square brackets are prefixed with the word "**input**". If there are internal events in the STIMULUS section, the word "**internal**" is prefixed to the name of the event. In the RESPONSE and OUTPUT sections, local names are suffixed with the words "**acceptance**", "**feedback**", "**rejection**", and "**request**" depending on the kind of response. "**feedback**" and "**acceptance**" are used to indicate a successful response from the system. "**rejection**" is used whenever the system rejects a stimulus (a possible reason might be that the stimulus is invalid). "**request**" is used when the system requests input or information from the user.

Descriptive names which improve the readability of the requirements are used for local names.

Guidelines For Writing Threads, Conditions, and Capabilities

Each specification unit may start with an OVERVIEW section. In a thread, the OVERVIEW section summarises the interaction between the inputs and the outputs. The section concisely describes the processing done if the inputs are validated and accepted. The section may end with a description of the context in which the thread exists and how the latter relates to other threads, conditions, and capabilities. The OVERVIEW section in a capability has the same format and contents as it does in a thread. The OVERVIEW section in a condition summarises the preconditions that are represented by the condition.

The overview is written in the active voice from the system's point of view. The OVERVIEW sections of threads and capabilities starts with the phrase "**This thread describes the processing performed upon receiving...**" and "**This capability describes the processing performed...**" respectively. In a condition, the OVERVIEW section starts with the phrase "**This condition describes...**".

Generally, the requirements are written in terms of what is externally visible and not in terms of internal processing. Therefore, no reference is made to data stores or internal databases. The requirements are written from a black box point of view. The requirements are written in the form "subject verb noun" or in the active voice in the future tense because it enables a simple sentence structure. The present tense is used when existing situations are described or definitions are specified.

The requirements in the REQUIREMENTS section must be written in a series of paragraphs where each paragraph is in the form *Upon the receipt of this class of stimulus, if 'condition A' or 'condition B', System shall return the corresponding class of response. System shall commit to this new system state.* Each paragraph is written based on the slightly redefined notion of a thread as a path connecting a *class* of stimuli to a *class* of responses.

Each paragraph must be complete, understood on its own, and must not depend or refer to a previous paragraph. Consequently, each paragraph may not use adverbs or phrases that have meaning based on what was previously described. For example, the following two examples are not acceptable.

```
Example 1:
   Paragraph 1 here {......}
   Next paragraph {Otherwise, .....}

Example 2:
  Paragraph 1 here {......}
  Next paragraph {'If both checks are positive', .....}
```

In both examples, the second paragraph refers to conditions described in the first paragraph. The second paragraph is incomplete without the first paragraph. The following concrete example is also incorrect because the requirements in the later paragraph rely on the fact that it was previously stated (paragraph 1) that the eligibility to register was specified in the condition "Check Eligibility To Register Condition".

```
    (Paragraph 1) 'If the student is not eligible to register for courses
     in [input session]' as specified in \Check Eligibility To Register
     Condition\, ......
     (Later Paragraph) 'If the student is not eligible to register', ....
```

There must not be any implicit sequentiality in the order in which the requirements are written as paragraphs. If an order is necessary, it must be explicitly stated.

Within each paragraph, each requirement is written in a separate sentence. In other words, requirements are not linked with the conjunction "and". For instance, writing *Telereg shall terminate the telephone connection and commit the system state to "IDLE"* *state* is wrong because each requirement should be written as a separate "shall" statement. That is, *Telereg shall terminate the telephone connection. Telereg shall commit the system state to "IDLE" state.*

Since natural language is inherently ambiguous, it is advantageous to enumerate conditions, actions, or responses whenever there is a list of them. Not only are itemized lists less ambiguous but they also stand out visually. For nested combinations of con-

System shall ... if all the following conditions are true/
  if at least one of the following condition is true:

a. condition a
b. All of the following conditions are true/At least one
   of the following conditions is true:
   1. condition b.1
   2. condition b.2
   3. ...
c. ...

Figure 6.23: Protocol for itemised lists

ditions, the protocol described in Figure 6.23 may be used. A similar protocol may be used for enumerating actions and responses. There is no sequential order implied in an enumerated list of conditions, actions or responses. If order is important, it must be explicitly stated. Enumerated lists are used because, unlike mathematical notation where potentially ambiguous expressions are disambiguated by associating various levels of precedence with the notations, the same cannot be done in natural language.

Requirements must be written so that they are as unambiguous as possible. For instance, the use of "it" must be avoided whenever possible. A requirement such as

If `it` *is in the "Waiting_for_identification_input" state, Telereg shall check...* is not as unambiguous as one where the pronoun `''it''` is substituted by "Telereg". Since the use of the slash (/) might be ambiguous in phrases such as "*valid/invalid*", it is not used except in "*and/or*" or in acronyms where its usage is less ambiguous.

If requirements refer to entities in the system state, their data dictionary entries, enclosed in angle brackets, are used in the text of the requirements. For instance, using "*<openings>*" instead of referring to the word "*openings*" in the requirements to indicate the number of openings in a section has two benefits. Not only can the meaning of the item be looked up in the data dictionary to understand its usage in this context but the vocabulary used in the requirements can be limited to using technical terms that are already understood and used in the problem domain. This reduces the chances of ambiguity. The requirements are then readable to the users and customers of the system who are familiar with the terminology used. Data dictionary items are used whenever requirements refer to the basic entities of the system.

Adjectives and other qualifiers which add no value to the adequacy of requirements must not be used. An adjective may be used if a distinction is to be made between two states of the same object. For instance, the qualifier "*validated*" is admissible for use in a term such as "*validated <student id>*".

Semantic checks on the inputs to a thread must be described explicitly. For instance, a requirement such as *Telereg shall check the validity of the format of the input...* is vague and uninformative whereas the same requirement written as *Telereg will check whether the [input student identification] can be found in the <student records>...* is valuable and informative. In this example, the requirement is written as a "will" statement because it is not externally testable.

If an input is not accepted for whatever reasons and no further processing is to be done, this must be explicitly stated because rejection of input may not necessarily imply termination of processing. Therefore the sentence "**No further processing will be done**" must be used whenever it is appropriate.

Mathematical algorithms are described with "will" statements rather than with "shall" statements. The requirements are written such that they specify how the outputs are derived from the inputs.

Since a thread may reference a capability, the thread should be written such that it can be read smoothly without having to reference the capability to understand the processing of the thread. A similar convention can be followed for conditions referenced by threads and capabilities and for capabilities referenced by capabilities and threads.

Finite State Machine Behaviour

Threads are purposely structured to stand alone and to be well defined [22]. Conditions may be included in both threads and capabilities. Capabilities can, in turn, be included in threads and capabilities. However, there is no description of how threads interrelate with one another. Although the finite state machine behaviour of systems may be strongly implied in threads and capabilities, the threads-based approach fails to explicitly give a cohesive view of systems with finite state machine behaviour [21].

Therefore, the threads-based technique is extended so that the modes of operation of a state may be captured in a descriptive section prior to the partitioning of the requirements into specification units. The system is described in terms of abstract or concrete states and the transitions from states to states upon the occurrence of events. This description gives an overview of how threads are related. Thus, in threads and capabilities, explicit references are made to the system being in certain states and the system committing to a new state when certain events occur. Events can be inputs or stimuli to a thread. Events may not arise due to results of checks on inputs being positive or negative. Threads may not trigger other threads by creating events whereby a variable is set to a particular value.

6.2.3 Threads-Based Development

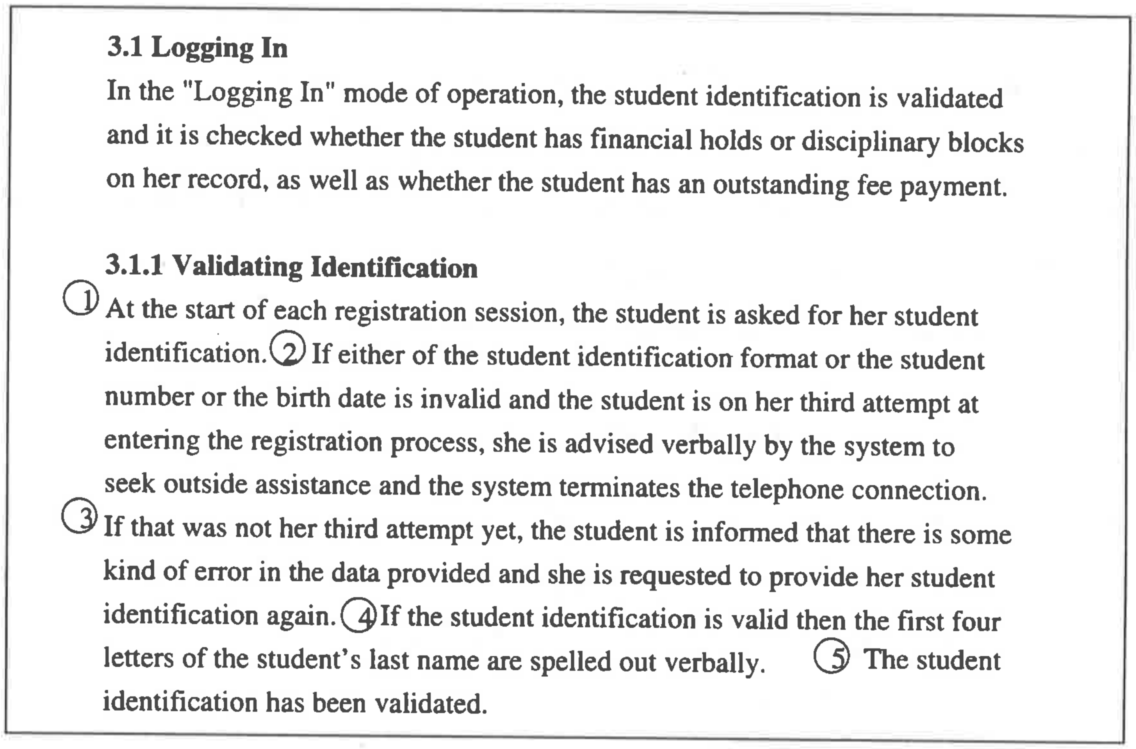
Since the SIS-Telereg system is of intermediate complexity, the allocation of its requirements into threads, capabilities and conditions was rather straightforward. The threads-based specification of the SIS-Telereg system was derived from the *ad hoc* natural language specification of the system. In a more sophisticated system, HCSD uses a multi-step threads-based development process [21]. First, each requirement is "binned" in a thread. After these preliminary allocations have been approved through a set of reviews, the STIMULUS, RESPONSE and OVERVIEW sections are specified. A second set of reviews is needed to approve that the stimuli and responses are correct and to establish guidelines for the general structure of the REQUIREMENTS section. After the requirements are specified, a last set of reviews involving the customer approve the allocation process and the specification effort. The specification style is also reviewed.

6.2.4 Tool Support for Threads-Based Technique

Since the threads-based technique is not a commonly used approach to specification, there are no commercial tools on the market to support it. However, HCSD has built its own tool support for internal use. The structure of the document facilitates any parsing process of the specification. Tool support may include a tool that checks that every data entity has an entry in the data dictionary and, based on attributes associated with each entry, other tools can automatically generate information about the specification document. Tools to compile a complete list of testable requirements can also be built.

6.3 Worked Example

Figures 6.24, 6.25, 6.26, and 6.27 describe part of the functionality of the SIS-Telereg system which has been specified *ad hoc* in natural language in Chapter 4. This might also be viewed as a sample of the kind of requirements description which a customer might provide. Figures 6.28 to 6.36 are the threads-based natural language counterpart of the *ad hoc* specification. It is assumed that the data dictionary names used in the threads-based specification have been previously established.



3.1 Logging In
In the "Logging In" mode of operation, the student identification is validated and it is checked whether the student has financial holds or disciplinary blocks on her record, as well as whether the student has an outstanding fee payment.

3.1.1 Validating Identification
① At the start of each registration session, the student is asked for her student identification. ② If either of the student identification format or the student number or the birth date is invalid and the student is on her third attempt at entering the registration process, she is advised verbally by the system to seek outside assistance and the system terminates the telephone connection. ③ If that was not her third attempt yet, the student is informed that there is some kind of error in the data provided and she is requested to provide her student identification again. ④ If the student identification is valid then the first four letters of the student's last name are spelled out verbally. ⑤ The student identification has been validated.

Figure 6.24: Natural language specification, part one

The OVERVIEW is the first section in a thread. By convention, the OVERVIEW starts with the phrase "*This thread describes the processing upon receiving...*". Sentence 1 in Figure 6.24 indicates that the input is a "*student identification*". This information is

3.1.2 Withholding student registration
If the student's record shows any disciplinary blocks or financial holds placed on it, then the student is not allowed to register. She is advised to contact the appropriate department or administrative office for more information. The student will be withheld from registering. This check is done when the student gives her student identification.

Figure 6.25: Natural language specification, part two

3.1.3 Checking for outstanding fee payment
The third check that is done upon a student providing her student identification is on her fee balance. If the student has an outstanding fee balance, she is reminded of it and the system will only allow the student to ask for transactions such as fee balance or section openings.

Figure 6.26: Natural language specification, part three

3.1.4 Exiting the "Logging In" mode of operation
The student will have to successfully pass the above three checks when she provides her student identification before she is allowed to proceed further in the registration process to the next mode operation, "Preprocessing". If any one of these three checks or combinations of checks fail, then the system will exit the current mode of operation.

Figure 6.27: Natural language specification, part four

added to the opening sentence in the OVERVIEW section. The validation done by the system as well as other kinds of processing done by the thread are described. Figure 6.28 is the OVERVIEW section of the thread which is titled "*Validate Identification Thread*" according to the format "*Verb Noun Thread*".

1.5.2 Validate Identification Thread
OVERVIEW:

This thread describes the processing performed upon receiving a student identification. Telereg will validate the input and if it is accepted, the first four letters of the student's last name are spelled out. It is also checked whether the student has financial holds or disciplinary blocks on her record, as well as whether the student has an outstanding fee payment.

Figure 6.28: Threads-based specification, overview

As the threads-based technique describes processing in terms of stimuli and responses, the next step is to identify the stimuli and responses. Sentence 1 in Figure 6.24 specifies that the input is a "*student identification*". Therefore, "*student identification*" is a stimulus to the thread. Figure 6.29 is the STIMULUS section of the thread. The word "*input*" is prefixed to the local name in the STIMULUS section.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input student identification] of the form:
a. <student_id>

Figure 6.29: Threads-based specification, stimulus

Sentences 2-5 in Figure 6.24 identify four system responses: a message that connection will be terminated, a rejection of the request, a request for the student identification and a last name message. These four messages are the first four data dictionary entries in the RESPONSE section (Figure 6.30), and they are each associated with a local name. Requirements in Figures 6.25 and 6.26 describe two other responses, namely a message that there is a hold or block on the student record and a message of outstanding fee payment. They are the last two entries in the RESPONSE section of Figure 6.30. In the RESPONSE SECTION, the local names each have one of the suffixes "*acceptance*", "*feedback*", "*rejection*" or "*request*".

While it is true that some administrative personnel may register on behalf of a student, the transactions only affect the student's records. Thus, the sources and destinations of stimuli and responses were not specified in the STIMULUS and RESPONSE sections because it is assumed that the student is the only source and destination of inputs and outputs respectively. This is also why each class of stimuli and responses only has one data dictionary entry listed under its associated local name. If the system were extended to allow for students to dial in through a modem to the system to register for courses, then an additional data entry "*Console* < *student_id* >" can be listed under the local name "[*input student identification*]" in Figure 6.29. The first data dictionary entry can then be changed to "*Phone* < *student_id* >".

After the stimuli and responses are identified, the requirements are specified. Requirement A in Figure 6.31 describes the initial system state for the registration process to start. Requirement A matches the requirement in Figure 6.24 (Section 3.1 Logging In) which describes the type of processing done in the "Logging In" mode of operation. Figure 6.31 assumes that in a previous section of the threads-based specification, the various states within the various modes of operations of the system have been detailed and that "Waiting_for_identification_input" has been described as the initial state within

RESPONSE:

Telereg shall return a [student identification acceptance] as follows:
a. <last_name_msg>

Telereg shall return an [invalid identification rejection] as follows:
a. <invalid_id_msg>

Telereg shall return an [identification request] as follows:
a. <request_id_msg>

Telereg shall return a [connection rejection] as follows:
a. <bye_msg>

Telereg shall return a [hold or block rejection] as follows:
a. <hold_or_block_msg>

Telereg shall return an [outstanding fee rejection] as follows:
a. <fee_kickout_msg>

Figure 6.30: Threads-based specification, response

the "logging in" mode of operation. Requirement A is a "will" statement because it is used to provide explanations.

REQUIREMENTS:

A If the system is in the "Waiting_for_identification_input" state, Telereg will check whether the [input student identification] can be found in the <student records>.

B Telereg shall return a [connection] rejection provided all of the following conditions are true:
a. 'The input cannot be found in the <student records>'.
b. 'It is the student's third attempt at entering her [input student identification]'.
Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

Figure 6.31: Threads-based specification, requirements

Requirement B in Figure 6.31 arises from sentence 2 of Figure 6.24. The conditions are itemised for clarity. Since no further processing is to be done after the telephone connection is terminated, it is explicitly stated. Since the requirements are to be written from an external point of view, the details of what constitute an invalid student identification (the student identification format or the student number or the birth date is invalid, as described in sentence 2 in Figure 6.24) are not described in the threads-based specification. The requirement of the telephone connection being terminated is a testable requirement and it is therefore described with a "shall" statement. The system transition to another state is meant to have a long term effect and be testable and hence it is described with a "commit" and a "shall". Local names and data dictionary entries are used whenever possible in the text of the requirements to limit its vocabulary and hence the possibility of ambiguity (note the usage of "*<student records>*" and "*[input student identification]*" in Requirement B in Figure 6.31 to limit the vocabulary used in the requirement).

Ⓒ Telereg shall return an [invalid identification rejection] and an [identification request] provided all of the following conditions are true:
a. 'The input cannot be found in the <student records>'.
b. 'It is not the student's third attempt yet at entering her [input student identification]'.
Telereg will stay in "Waiting_for_identification_input" state.

Figure 6.32: Threads-based specification, requirements (cont.)

Requirement C in Figure 6.32 is the threads-based technique counterpart of requirement 3 (Figure 6.24) in the *ad hoc* approach. The conditional statement used in sentence 3 of Figure 6.24 ("If that was not her third attempt yet, the student is informed that there is some kind of error in the data provided and she is requested to provide her student identification again.") relates to the previous sentence in the same paragraph. In the threads-based technique, however, each paragraph must stand alone and therefore condition a is explicitly stated so that the paragraph (requirement C) can be read irrespective of requirement B. The fact that the system stays in the same state is described with a "will" statement because it is only a clarifying note.

The requirement described in Figure 6.25 is translated into paragraph D of Figure 6.33. Condition a is necessary otherwise no check would be done on the student's record. Similarly, requirement E is the threads-based counterpart of the requirement in Figure 6.26. The "will" statement in requirement E hints at the consequences of the conditions being true.

Paragraph F in Figure 6.35 is a combination of information from sentence 4 in Figure

Telereg shall return a [hold or block rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

D b. 'There are <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

The telephone connection shall be terminated. No further processing will be done. Telereg shall commit the system state to "IDLE" state.

Figure 6.33: Threads-based specification, requirements (cont.)

Telereg shall return an [outstanding fee rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

E b. 'The student has an <outstanding fee balance>' as specified in the \Check Outstanding Fee Assessment Condition\.

Telereg will then only allow the student to ask for transactions such as current fee assessment and inquiry on section openings.

Figure 6.34: Threads-based specification, requirements (cont.)

Telereg shall return a [student identification acceptance] if all of the following conditions are true:

F a. 'The input can be found in the <student records>'.

b. 'There are no <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

Telereg shall commit to the "Waiting_for_session_input" state.

Figure 6.35: Threads-based specification, requirements (cont.)

6.24 and Figures 6.25 and 6.26. The "Waiting_for_session_input" state is assumed to be the first state in the "Preprocessing" mode of operation described in Figure 6.27.

Figure 6.36 gives a sample of the data dictionary entries of the threads-based specification. The data dictionary entries are defined in terms of comments about their contents and usage. In complex systems, more structure must be added to the usage of the DD.

6.4 Strengths and Weaknesses of the Threads-Based Technique

This section presents an evaluation of the threads-based technique.

6.4.1 Strengths

1. **A threads-based specification is potentially less ambiguous than an *ad hoc* specification.**

   The threads-based specification technique offers the advantage of writing requirements in a less *ad hoc* manner than the natural language approach. The threads-based technique is potentially less ambiguous than the *ad hoc* natural language approach because of the added structure and writing conventions imposed by the technique. For example, the usage of enumerated lists for actions, conditions and responses and limited vocabulary with keywords and key phrases has the advantage of decreasing the chances of ambiguous requirements.

2. **The threads-based technique facilitates the specification process.**

   The threads-based approach provides rules of convention and a writing style that give the functional requirements structure. It provides a systematic and thus standard way of writing requirements. This is especially advantageous when a team of authors specify the requirements of a large system. Since the specification document is the base document for all the ensuing development phases in the software

```
<bye_msg> =
    * This is a message that is given by the system if the student fails to enter
    a valid student identification within three attempts. The system will not
    allow the student to proceed. *

<disciplinary blocks> =
    * These are restrictions that are placed on a student's record to prevent
    her to register. These restrictions arise because of academic reasons
    such as failure to keep in good standing and so on. *

<fee_kickout_msg> =
    * This message is given by the system if it finds that the student who is
    attempting to register has an outstanding financial account. The student
    will only be able to ask for a limited set of transaction requests. *

.

.

.

<student_id> =
    student_number + birth_date
    * The identification that  a student provides in order to register. *

<student records> =
    * The complete list of files that the system has on the students at the
    university. *
```

Figure 6.36: Data Dictionary entries, a sample

life cycle, a consistently styled document facilitates the design and test engineers' task of understanding the system.

3. **The threads-based technique can aid in validating requirements.**
   The threads-based technique requires that each requirement be specified in a separate paragraph and that each paragraph be complete and understood in isolation. These rules increase the chances of finding loopholes in the requirements and enables an informal validation of requirements. Indeed, in the SIS-Telereg system, numerous loopholes were uncovered in the *ad hoc* specification document as a result of writing its threads-based counterpart. To be specific, it was discovered that the *ad hoc* specification (given in Appendix A) failed to decrement the number of free transactions in several functions. Also, certain functions could only be performed if there was no outstanding fee balance on the student's account, but the *ad hoc* specification failed to state this requirement.

4. **The threads-based technique discourages design.**
   A threads-based specification is structured so that it is completely decoupled from design. This is accomplished by imposing that the document be specified in terms of what is externally visible. Thus, any decisions made in the design phase do not affect the requirement specification. This limits any maintenance problems in keeping documents consistent across the various development stages.

5. **The threads-based technique facilitates generation of test cases.**
   A threads-based specification is structured so that it defines a system by explaining the characteristics the system must provide to an outside user. Together with the added structure, the black box description of the requirements makes the specification a testable document in that it facilitates the task of generating test cases from the specification. Also, the threads-based approach mandates that it is always

shown where data comes from with the key phrase "*included in [local name]*"; this can facilitate the testing task. "The strength of the stimulus-response model is in charting the procedure or scenario that transforms a stimulus into an eventual system response" [12].

6. **The threads-based technique facilitates information extraction from the specification.**

   While there is no tool support to check the internal consistency of the specification, there is tool support to parse threads-based text and extract information from it using intelligent analysis. The notational conventions used in the technique facilitate the extraction of information from the document [20].

7. **The threads-based technique facilitates system testing.**

   In system integration testing, the threads-based approach lends itself to the builds approach of integrating the major components of a system. "Like the threads technique, the builds approach is founded upon the demonstration of a functional stimulus-response scenario" [12]. The builds approach enables successive partial integration of the system early during system development. This helps boost up the customer's confidence in the progress and achievement of the system product. The system developers can also get satisfaction from seeing some results to their efforts.

6.4.2 Weaknesses

1. **The use of natural language in the threads-based technique causes ambiguity.**

   The threads-based technique uses natural language as a medium to specify requirements and in spite of the structure and writing style imposed by the approach, *Telereg shall return an [invalid request rejection] if the request is invalid and the connection is to be terminated as specified in \Invalid Request Rejection Condition\......*

Figure 6.37: Requirement in "Check Validity of Requests Capability"

natural language is inherently ambiguous and gives rise to ambiguous requirements because of its impreciseness. Consider the first paragraph of the REQUIREMENTS section in the SIS-Telereg capability "Check Validity of Requests Capability" described in Figure 6.37. The requirement is ambiguous and can be interpreted in

(a) *Telereg shall return an [invalid request rejection] if both the following conditions are true:*

  a. *the request is invalid*

  b. *the connection is to be terminated*

  and both these conditions are specified in \Invalid Request Rejection Condition\.

(b) *Telereg shall return an [invalid request rejection] if both the following conditions are true:*

  a. *the request is invalid*

  b. *the connection is to be terminated as specified in \Invalid Request Rejection Condition*

(c) The requirements are really two separate requirements:

  a. *Telereg shall return an [invalid request rejection] if the request is invalid*

  b. *the connection is to be terminated as specified in \Invalid Request Rejection Condition*

Figure 6.38: Interpretations of requirement in "Check Validity of Requests Capability"

at least three different and plausible ways (See Figure 6.38). However, the intended meaning is none of the three ways described in Figure 6.38; it is actually a refinement of the first interpretation with the two conditions merged together as a single condition with the name "*the request is invalid and the connection is to be terminated*" being descriptive of what the condition "Invalid Request Rejection Condition" specifies. Since a single condition was specified, it was not described in an itemised list. But even if the requirement had consisted of two separate conditions, using natural language does not help specifying them in textual form without there being any ambiguity.

Telereg shall return an [invalid request rejection] if 'the request is invalid and the connection is to be terminated' as specified in \Invalid Request Rejection Condition\......

Figure 6.39: Changed requirement in "Check Validity of Requests Capability"

In this case, using single quotes to delineate system states and condition names eliminated the ambiguity (See Figure 6.39). Other mechanisms could have been used as well. While precautions may be taken to avoid ambiguity, using natural language inherently causes ambiguity.

2. **The threads-based technique does not encourage the specification of invariants.**
   Invariants are important properties of a system to specify. Unfortunately, the threads-based technique does not encourage the specification of invariants. From Figure 6.40, the third paragraph of requirements for the capability "Check Validity of Requests Capability", it could be asked whether Telereg can return both a "[free transactions left feedback]" and a "[maximum charge fee rejection]" if the registration request is valid. As the requirements are written, the answer is positive if the student has more than zero but ten or fewer free transactions and the

If the registration request is valid,

a. *Telereg shall return a [free transactions left feedback] if 'the student has more than zero but fewer than ten <free transactions> left to use up'.*

b. *If 'the <maximum charge fee> is exceeded', Telereg shall return a [maximum charge fee rejection]. Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.*

c. *If 'the <maximum charge fee> is not exceeded' and 'the student is to be charged a <transaction fee>' as specified in \Charge Fee For Transaction If Necessary Condition\, Telereg shall charge the student a <transaction fee>. Telereg shall return a [fee charge feedback] to the student. Telereg shall commit the student's number of <free transactions> to one less.*

Figure 6.40: More requirements from "Check Validity of Requests Capability"

maximum charge fee is exceeded. But a closer examination of the data dictionary entries "<free transactions>" and "<maximum charge fee>" would indicate that "<maximum charge fee>" is at zero and increases only when the number of "<free transactions>" becomes zero. Therefore, Telereg can never return both a "[free transactions left feedback]" and a "[maximum charge fee rejection]" at the same time. Hence, it is only with domain knowledge that one would have been able to answer the question correctly. The fact that the "<maximum charge fee>" cannot be exceeded as long as there are any "<free transactions>" is therefore an invariant of the system that would have added value to the specification were there a means to force us to specify it in the threads-based approach.

3. **The threads-based approach does not explicitly capture the finite state machine behaviour of systems.**

   The threads-based approach was designed to specify what is externally testable; hence its emphasis on explicitly highlighting the external stimuli and responses.

The technique discourages internal interaction in that internal responses cannot be created but external responses can be observed. Thus, the threads-based approach does not lend itself to specifying sequential behaviour. Any finite state machine behaviour is specified separately in natural language.

6.5 Conclusion

This section evaluates the threads-based approach as an informal specification technique.

6.5.1 Remarks About the Threads-Based Technique

Obviously, the conventions of this approach will vary from problem domain to problem domain. As the complexity of the system grows or as the size of the system grows, the number of conventions may be increased to ensure a systematic and uniform codification of the requirements. The conventions may also change as the focus of the system or customer requirements vary. In other words, the list of conventions may be domain specific.

In a system as complex as the one undertaken by HCSD, many more conventions than the oncs described previously are standardised to facilitate the systematic and uniform derivation of the requirements. Large numbers of documents are referenced in the requirements of a large system. These documents may be referred by name and a document number previously assigned to it. The specification document must include a list of the reference documents at the end of the specification document. Also, with complex system functionalities, the requirements may be divided into bigger functional chunks. HCSD uses headers that encapsulate some functionalities that are then described in threads, capabilities, and conditions. The headers describe the relationship between the threads, capabilities and conditions and their requirements apply to all the functionality in the component specification units [16]. In the SIS-Telereg system, headers were not needed because of the simplicity of the system.

6.5.2 Evaluation of the Threads-Based Technique as an Informal Technique

The threads-based technique can produce less ambiguous requirement statements than the *ad hoc* informal approach. The document organisation and writing style enable a systematic way to specify requirements uniformly. The threads approach also facilitates the informal validation of requirements by requiring that each paragraph of requirement be complete. The threads-based technique advocates an external black box view of the system assuming that the requirements are decoupled from design and the specification document is testable. However, the use of natural language as the medium of communication causes most of the downfalls of this technique: ambiguity and impreciseness. But, overall, the threads-based technique is a useful approach to specification. If an organization has reservations investing in more formal methods than the *ad hoc* approach, the threads-based approach is a good trade-off by allowing the use of natural language in a structured format.

Chapter 7

Formal Specification Language

7.1 Introduction

Formal approaches to specification are becoming increasingly popular. System failures have pointed to a need to write requirements formally so that they can be precisely specified and various analysis such as formal validation and formal verification can be done on the requirements. The formal language Z (pronounced "zed") is one of the popular formal specification notations which has been used successfully in several industrial applications. Z is quite popular in the UK where there is considerable interest by industry and government in formal methods. Also, Z is applicable to the SIS-Telereg system.

Z is a machine readable mathematical notation based on set theory and first order logic. Z is used to build a mathematical model of a system whereby the problem domain consists of data objects and the relationships between the various data objects in the system are described by the requirements. The data objects are made of elements called basic types. The building block for a basic type is sets and functions. Tools are available to provide automatic checking for type consistency across the specification.

A specification written in Z is a mixture of formal, mathematical statements and informal text. The formal mathematical statements give a precise description of the system in terms of operations performed on the data objects by specifying the system state before processing an event and after processing the event. The informal text describes in natural language the meaning of the mathematical statements to make the specification more readable. The formal statements are grouped into packaging mechanisms called *schemas.* Each schema consists of a signature part that declares the variables used in the schema's operation and a body part that describes the operations. The schemas are abstraction modules and they are combined to produce a specification of the system.

Section 7.2 defines the specification language Z, and describes the various notational tools and rules of conventions used in Z. In addition, the section describes how a Z specification can be developed in general and, more specifically, from a threads-based specification document. Tool support for Z is described as well. Section 7.3 describes how the Z notation is used to derive part of the specification of the SIS-Telereg system from its threads-based counterpart. Section 7.4 evaluates Z. Finally, Section 7.5 concludes with some observations about Z and about formal specification techniques in general.

7.2 Z

Z is a mathematical specification language based on set theory and first order logic. It was developed by the Programming Research Group at the Oxford University Computing Laboratory. It is now used in software and hardware development industries in the UK and the US. Z is a popular notation in both academic and industrial circles. Z has made some significant contributions on projects where it has been used. It has been used successfully to specify requirements in several industrial projects. Craigen *et al.* [7] [8] report the successes of IBM's CICS (Customer Information Control System), Tektronik, and INMOS transputer in their attempt to use Z . The popularity of Z is such that it is beginning to appear in undergraduate software engineering textbooks, e.g. Sommerville [29], as an introduction to formal specification.

Z uses mathematical expressions of sets and logic to specify what a system is required to do. Z is used to specify a mathematical model of the system which describes its behaviour. Z views the application domain as consisting of primitive concepts that can either be kinds of data objects (primitive data objects) or operations that only involve primitive data objects. The specification is built on top of these primitive concepts. These primitive concepts define the level of abstraction of the Z specification. The system is considered to be in one of a set of possible states and the model describes how the system makes a transition from one state to another upon the occurrence of an event. An event may cause the system to perform an operation that will change the system state.

Set theory is used to describe the primitive data objects in the system, the possible system states, the relationships between system data objects, and the operations on the data objects. First order logic is used to link occurrences of events to the resulting operations performed. Therefore, a model of a system is described in terms of logic expressions that relate sets whereby the sets describe the data objects of the model, their relationships and operations performed on them. A type is associated with each data object. A type is also associated with each relationship between data objects.

A Z specification is structured such that it can be developed incrementally just like a program is structured in terms of functions that refer to other functions. A Z specification consists of mathematical statements that are grouped together in packaging mechanisms called **schemas** whereby a schema may be expressed in terms of other schemas. A schema is a unit of specification that describes requirements as a relationship between the state of the system before processing an event and the state of the system after processing the event. The operation described by a schema causes the change of state. Each schema must be a unit of specification that can be understood in isolation. Each schema is accompanied with corresponding explanatory text that explains the mathematical statements. Schemas describe the dynamic behaviour of a system. The dynamic behaviour consists of operations performed by the system, the relationships between inputs and outputs, and the state changes in the system. There is no implicit sequentiality in the order in which requirements are written. There is a single level of abstraction in a Z specification in the sense that there are multiple levels of abstraction in a Structured Analysis specification (chapter 5).

A Z specification must also describe the static properties of a system. The static properties include the possible system states and the invariant relationships that are maintained as the system moves from state to state. Each type of static property is specified with a slightly different notation; each notation is called a **paragraph**. The static properties are described in paragraphs called **basic types, defined types, abbreviation types** and **axiomatic descriptions**. Schemas are also a type of paragraph and they are used to describe the dynamic nature of a system. Paragraphs, set expressions and logic are the notational tools in Z.

7.2.1 Notation Used in Z

Basic Types

The basic types of a specification are the fundamental concepts in a system. They are the primitive data objects in the system. "The basic types are the building blocks of the Z type system" [24]. Basic types are enclosed in square brackets in the form

$$[typename, typename, ..., typename]$$

where *typename* is a basic type name. For instance, in a student information system, entities such as "*students*" and "*student records*" are assumed to be primitive concepts. Therefore, some basic types of a student information system are

$$[STUDENT, RECORD]$$

Defined Types

Defined types (also called data type definitions) in Z are analogous to enumerated types in programming languages. They define sets with a small number of members. These sets are represented in the form

$$Name ::= Element1 \mid Element2 \mid ... \mid Elementn$$

The name of the set is associated with its list of members by the symbol (::=). The list of members are separated by a vertical bar (|). Elements of defined types are distinct from one another and there may not be other elements for that type. For example, if a student can be classified either as a freshman, sophomore, junior, or senior, a defined type "*classification*" can be defined as follows:

$$classification ::= freshman \mid sophomore \mid junior \mid senior$$

Abbreviation Definitions

Abbreviation definitions define global constants. Global constants are introduced with the double equality sign (==). Global constants are of the form:

$$Name == Expression$$

where *Expression* is an expression that associates a predetermined value to the identifier *Name*. The following example is an abbreviation definition which specifies that a student may enroll in a maximum of eighteen credits:

$$MaxCredits == 18$$

Global constants may be used anywhere throughout a specification.

Axiomatic Descriptions

A specification may declare global variables whose values are constrained. Axiomatic descriptions are therefore used to "introduce variables along with predicates giving further information" [24]. Global variables can be referenced throughout a specification. Axiomatic descriptions can be of the following two forms:

Declarations

Predicates

Declarations

If "*numbercredits*" is the number of credits that a student is enrolled in, then the paragraph below specifies that a student may not be enrolled in more than eighteen credits.

$numbercredits : \mathbb{N}$

$numbercredits \leq 18$

Schemas

Schemas are the major building blocks for constructing a Z specification. A schema is a unit of specification which describes operations on the data objects of a system. Schemas may have a linear or a graphical representation. The following example illustrates a graphical schema.

$x : \mathbb{N}$

$y : \mathbb{P}\mathbb{N}$

$x \in y$

A graphical schema is a box consisting of two sections. The section above the middle line describes the signature or declaration of the schema; in other words, it introduces variables and their associated types. The signature section corresponds to declarations in a programming language. Declarations in the signature section may either be written on separate lines or separated by a semicolon (;) if they are written on the same line.

The section below the middle line is the predicate section. It specifies the relationships between the variables declared in the schema, other schemas and global variables. If the predicates are written on the same line, they are delineated by a semicolon (;). The predicate for a schema is the conjunct of all the predicates in the predicate section. Graphical schemas are used because they visually stand out from their corresponding explanatory text.

The linear form of the above graphical schema is as follows:

$$[x : \mathbb{N};\ y : \mathbb{P}\,\mathbb{N} \mid x \in y]$$

The schema is enclosed in square brackets ([]). The signature appears before the predicate and they are separated by a vertical bar (|). Linear and graphical schemas are semantically equivalent. Linear schemas are used to save space or for simple definitions.

Schemas are named so that they can be referred by other schemas. In a graphical schema, the name appears on the line at the top of the box. In the linear form, the name is associated with the definition of the schema by the equality symbol ($\widehat{=}$). For instance, if "ElementOf" is the name of the schemas described above, then the schemas would be as follows:

```
┌─ ElementOf ─────────────────────────────
│ x : ℕ
│
│ y : ℙℕ
├──────────
│ x ∈ y
└─────────────────────────────────────────
```

and

$$ElementOf \mathrel{\widehat{=}} [x : \mathbb{N};\; y : \mathbb{P}\mathbb{N} \mid x \in y]$$

Generally, schemas are of the form

```
┌─ Name ──────────────────────────────────
│ Signature
├──────────
│ Predicates
└─────────────────────────────────────────
```

and

$$Name \mathrel{\widehat{=}} [Signature \mid Predicates]$$

Generic Definitions

The notation for a generic definition is a box like the one used in a schema except that the top line is a double line. Generic definitions are used to parameterise a schema over any type and they are of the form:

```
╒═[Parameters]════════════════════════════
│ Declarations
├──────────
│ Predicates
└─────────────────────────────────────────
```

The generic type of parameters that are to be used in the definition are indicated at the top line.

Consider the following generic definition:

$$
\begin{array}{|l}
[T] \\
\hline
\_\ Union\ \_ : \mathbb{P}\, T \times \mathbb{P}\, T \rightarrow \mathbb{P}\, T \\
\hline
\forall X, Y : \mathbb{P}\, T \bullet X\ Union\ Y \Leftrightarrow \\
\qquad \exists p : T \mid p \in X \vee p \in Y \\
\hline
\end{array}
$$

The notation "_ *Union* _" indicates that "*Union*" is an infix operator that takes two arguments of the same type.

7.2.2 Schema Calculus

Schema calculus describes the operations that can be performed on schemas, for example logical conjunction of schemas. These operations allow schemas to be expressed in terms of other defined schemas, thereby enabling a modular presentation of the specification.

Schema Inclusion

To allow a schema to refer to others, the referred schema is included in the signature section of the schema that references it. This is equivalent to the union of the signatures of both schemas and the conjunct of their predicate sections. It is assumed that the same name is used to refer to the same data object. Duplicated names are replaced by a single instance of the name provided they are of the same type.

The following trivial example illustrates the semantics of schema inclusion. If the schema "*AlsoElementOf*" is defined as:

$$\begin{array}{|l}
\hline
\mathit{AlsoElementOf} \\
x : \mathbb{N} \\
z : \mathbb{P}\mathbb{N} \\
\hline
x \in z \\
\hline
\end{array}$$

then including the previously defined schema " *ElementOf*" in the schema " *AlsoElementOf*" as follows:

$$\begin{array}{|l}
\hline
\mathit{AlsoElementOf} \\
x : \mathbb{N} \\
z : \mathbb{P}\mathbb{N} \\
\mathit{ElementOf} \\
\hline
x \in z \\
\hline
\end{array}$$

is equivalent to

$$\begin{array}{|l}
\hline
\mathit{AlsoElementOf} \\
x : \mathbb{N} \\
y : \mathbb{P}\mathbb{N} \\
z : \mathbb{P}\mathbb{N} \\
\hline
x \in y \\
x \in z \\
\hline
\end{array}$$

Schema Disjunction

A schema may be defined as the disjunct of two other schemas with the schema operator $\vee$. For instance, consider a schema P defined as $P \mathrel{\widehat{=}} A \vee B$ where A and B are predefined schemas. Assuming that the same name is not used for two different variables in schemas A and B and the types of shared variables agree, the declaration section of P is the union of the declaration parts of A and B. The predicate section of P is the disjunct of the predicates of A and B.

Schema Conjunction

A schema may be defined as the conjunct ($\wedge$) of two other schemas. Schema conjunction is analogous to schema disjunction except that the predicate section of the resulting schema is the conjunct of the predicates of the component schemas.

Schema Negation

A schema may be negated with the negation operator $\neg$. For any schema P, the schema $\neg\ P$ is obtained by keeping the signature section of P and negating the predicate section.

Schema Composition

The schema operator $\fatsemi$ is used to specify an operation as a composition of operations. Therefore a definition such as

$$M \mathrel{\widehat{=}} P \fatsemi Q$$

means that operation M is the composition of operations P and Q provided there exists a state S2 such that if operation P causes a change of state from some state S1 to S2, then operation Q causes a change of state from S2 to some state S3. Here, operation is synonymous to schema since a schema describes an operation.

Schema Extension

The signatures and predicates of schemas can be extended. A new declaration may be added to a signature by following the schema name by a semicolon and the new declaration. For instance, to add the declaration "$p : \mathbb{N}$" to the previously defined schema " *ElementOf*", a new schema "*NewElementOf*" is defined as

$$NewElementOf \mathrel{\widehat{=}} [ElementOf;\ p : \mathbb{N}]$$

Similarly, to extend the predicate section of a schema, the new predicate is concatenated to the schema name with the symbol '|'. Therefore, to add the predicate "$p \in y$" to schema "*NewElementOf*", the new schema "*NewerElementOf*" is defined as follows:

$$NewerElementOf \mathrel{\widehat{=}} [NewElementOf \mid p \in y]$$

7.2.3 Conventions for Z

Basic types are written in capital letters. Conventions may also be used to distinguish between defined types, global variables, and global constants by capitalising the first letter of a name or the first letter of the words of a name and so on. These conventions are left to the specifier's discretion.

Some of the variables in the signature section of a schema may be input variables and output variables. Input variables are variables whose values will be used in the operation described in the schema while output variables represent variables that change values after an event is processed in the schema. Input variables have the symbol "?" as their final character while output variables are suffixed with the exclamation mark "!". Consider the following example: a student provides a section number that she would like added to her schedule. The registration system gives her back a message indicating the status of the transaction. The schema describing the operation of adding a section may have variables "*section*" and "*response*". In the signature section of the schema, the variables are written as "*section?*" and "*response!*".

The value of objects before processing are represented by plain variable names and their corresponding values after processing are represented by dashed (primed) variable names. In the above example, after the add operation, the student's schedule would have changed. If the variable "*schedule*" is used before the addition operation is performed, then the variable "*schedule′*" is used to reflect the change to the value of the entity "*schedule*" after processing. Priming variables is known as "variable decoration".

A schema may also be primed. This means that all the variable names and relationship names in both the signature and the predicate sections of the schema are primed. Decoration of schemas enables us to specify that an operation starts from a valid state and results in a valid state after the operation. When both a schema S and its primed counterpart S' are declared in the signature of a schema, they can be replaced by the semantically equivalent "Δ S". The symbol "Δ" is used to signify a change of state.

Therefore, a schema name is prefixed with the symbol "Δ" to indicate that the general properties of the schema are unaffected by a series of events. The schema described with the "Δ" symbol prefixed to its schema name describes invariant properties of the system. The schema named "Δ*ElementOf*" can be used as follows to describe some invariant properties of a system:

```
┌─ ΔElementOf ──────────────────────────
│ ElementOf
│
│ ElementOf′
└───────────────────────────────────────
```

On the other hand, some operations may not cause any change in the system state. The symbol "Ξ" is prefixed to a schema name to denote that there is no change of state.

This is equivalent to having an unprimed and primed copy of the declarations of a schema with the predicate section of the new schema having all unprimed variables being set to be equal to their corresponding primed variables.

7.2.4 Z Development

The Z notation does not inherently guarantee that all the parts of the system which need to be modelled will be specified. Therefore, guidelines are set to facilitate the specification process to ensure that all the requirements are described in a comprehensive manner. Potter *et al.* [24] describe guidelines for the overall structure of specification documents based on ideas developed by the collaborative efforts of IBM Hursley and Oxford University Programming Research Group; these guidelines are described here. These guidelines propose a certain organization to a Z specification; they help maintain the same style of specification over projects.

Overall, the document is structured such that each specification paragraph is accompanied with natural language text, which uses technical jargon of the application domain, to explain the mathematical expressions. An overview of the function and scope of the system is given at the beginning of the specification document. Throughout the document, data objects are declared before they are referred to. While the various parts of a specification are presented in a certain order, they may not necessarily be developed in that order. This process of developing a Z specification is standard to when it is derived from an *ad hoc* natural language specification.

The specification document is organised with the following sections included in the order listed: declaration of basic sets and global variables, general theory about the system, abstract states of the system, initialisations of the system, successful operations performed by the system, preconditions of defined operations, operations under error conditions, and summary and index of specification.

The document first defines the static properties of the system and then specifies the dynamic properties of the system. Before the document can define the bulk of the specification, which is the operations in the system, the primitive concepts in the system have to be established. Therefore, the different kinds of data objects that constitute the system (basic sets) are first identified. Each kind of data object represents a distinct type. Defined types are also defined. Then, the things that are global throughout the model are defined. Global variables are also declared. Next, theories that are needed by schemas later in the specification are defined. The major components of the model that may change value over time are identified. The operations performed by the system are described when all preconditions are satisfied. They make up the bulk of the specification. Preconditions are calculated to determine whether an operation is feasible and to discover the range of error conditions that may arise. "The preconditions of the partial operations indicate when those operations may be applied successfully, and thus the negations of those preconditions indicate the range of cases which have to be treated as errors and handled accordingly, if we want to ensure that the system can respond appropriately under all circumstances." [24]. Finally, for large specification documents of non-trivial systems, an index listing variables and schema names and the pages on which they are referenced is included at the end.

Z Development from a Threads-Based Specification

In spite of guidelines given in Z textbooks, including the ones given in Potter *et al.* [24], about how to organise Z specifications, it is hard to get started in developing a Z specification. The guidelines are not sufficient to enable a systematic derivation of a Z specification. For instance, issues such as the partitioning of requirements into schemas that are small enough to be readable and big enough to be non-trivial are not addressed.

A preliminary research investigation [19] in the possible application of formal methods to large systems revealed that the threads-based specification lends itself rather naturally to be formalized in Z. It resulted in some guidelines to enable the systematic translation of a Z specification from a threads-based specification. The Z specification of the SIS-Telereg system was derived from the threads-based specification and by following the general organisation proposed by Potter *et al.* [24].

The specification style of Z approach closely matches that of the threads-based approach. Both approaches have a similar document structure. In both approaches, requirements are described in terms of operations on entities in the system whereby each operation is encapsulated in a specification unit. Both encourage the specification of requirements in terms of what is externally observable. In the threads-based approach, the specification unit describes the processing done upon the occurrence of an external stimulus and the resulting external response. In Z, the specification unit describes the system state before processing an event and the system state after processing the event. Outputs are described as well in Z.

7.2.5 Tool Support for Z

Since Z specifications consist of non-ASCII symbols, they have to be written in an encoded form so that they are machine readable. Formatting programs such as LaTeX can be used to allow document editing of Z specifications. The following example shows how a requirement in Z can be encoded in LaTeX.

```
\begin{schema}{VIT\_input\_student\_identification}
        Telereg\_State
\also
        id: SId
\also
```

```
        input? : Stimuli\_Or\_Responses
\where
        input? = student\_id(id)
\end{schema}%VIT\_input\_student\_identification%
```

In addition, computer-based tools are available to type-check Z specifications, for example, "Fuzz" [30]. Manual checking is required to maintain consistency between the mathematical statements and the informal text.

7.3 Worked Example

This section describes how part of the threads-based specification (see Chapter 6) for the SIS-Telereg system can be used to derive a Z specification. As an illustration, the thread "Validate Identification Thread" is specified in Z. Before the operations described by the thread can be specified in Z, the static attributes of the system must be defined. Following the guidelines proposed by Potter *et al.*, the basic sets are first identified. The only basic type that is needed for this thread is

$$[SId]$$

where "SId" defines a set of all possible student identifications as a basic type in the system. Therefore, the global set "$Students : \mathbb{P}\, SId$" is declared to define the set of all students who are enrolled at the university. A schema called "Telereg_State" is also defined as:

$$\begin{array}{|l}
\textit{Telereg\_State} \\
\hline
Blocks : \mathbb{P}\, SId \\
Holds : \mathbb{P}\, SId \\
Outstandings : \mathbb{P}\, SId \\
MAX\_ID\_COUNT : \mathbb{N} \\
\hline
\end{array}$$

The sets "Blocks", "Holds" and "Outstandings" define the sets of students who have disciplinary blocks, financial holds, and outstanding fee balances respectively. "MAX_ID_CO-UNT" is also declared as a global variable which defines the maximum number of times a student identification can be inputted by a student in a registration session. Other static attributes will be defined as they are needed.

Each specification unit in the threads-based specification is kept as a specification unit in Z. Each such unit is made of schemas and text organized in the following sections: History, Dependencies, Overview, Stimuli, Responses, Conditions and Modes, Actions, and Requirements. Threads have all the listed sections. Capabilities may have all but the "Stimuli" section. Conditions only have the "History", "Overview" and "Conditions and Modes" sections.

The Z specification starts with a title identifying the thread which is being specified. Next, the "History" section is used to record changes made to the document by the specifier. The "Dependencies" section lists the capabilities and conditions which are referenced in a specification unit. Figure 7.41 describes the two conditions which are referred in the thread. The list of dependencies is specified only for documentation purposes. Next, the OVERVIEW section of the thread is copied word for word in the "Overview" section in the Z specification.

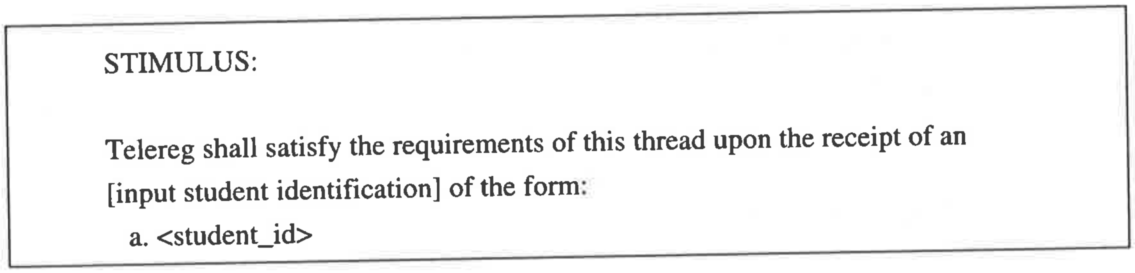
As a matter of convention, in the rest of the specification, the graphical schema is used when local variables have to be declared in the signature section. Otherwise, if only

Dependencies

1. WSRC_holds_or_blocks_condition as specified in \Withhold Student Registration Condition\
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Figure 7.41: Z specification, Dependencies section

a change in the system state, "Δ *Telereg_State*", is to be declared in the signature section, the linear schema is used. Each schema is accompanied with natural language text to describe its mathematical components. Each schema name is prefixed with "VIT" which is an acronym for the name of the specified thread: "Validate Identification Thread".


STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input student identification] of the form:

a. <student_id>

Figure 7.42: Threads-based specification, stimulus

The STIMULUS and RESPONSE sections in the thread correspond to the "Stimuli" and "Responses" sections in the Z specification. Figures 7.42 and 7.43 are the threads counterpart of the Z specification described in Figures 7.44 and 7.45. In Z, the stimuli and responses are modelled as elements of a defined set representing all the stimuli and responses in the system. The data dictionary entries in the thread are used as elements

RESPONSE:

Telereg shall return a [student identification acceptance] as follows:
a. <last_name_msg>

Telereg shall return an [invalid identification rejection] as follows:
a. <invalid_id_msg>

Telereg shall return an [identification request] as follows:
a. <request_id_msg>

Telereg shall return a [connection rejection] as follows:
a. <bye_msg>

Telereg shall return a [hold or block rejection] as follows:
a. <hold_or_block_msg>

Telereg shall return an [outstanding fee rejection] as follows:
a. <fee_kickout_msg>

Figure 7.43: Threads-based specification, response

of a defined type "*Stimuli_Or_Responses*" defined as:

$$
\begin{array}{l}
\mathit{Stimuli\_Or\_Responses} ::= \\
\quad \mathit{bye\_msg} \mid \\
\quad \mathit{fee\_kickout\_msg} \mid \\
\quad \mathit{hold\_or\_block\_msg} \mid \\
\quad \mathit{invalid\_id\_msg} \mid \\
\quad \mathit{last\_name\_msg} \mid \\
\quad \mathit{request\_id\_msg} \mid \\
\quad \mathit{student\_id}\langle\!\langle \mathit{SId} \rangle\!\rangle
\end{array}
$$

The element "*student_id*" has a parameter of type "*SId*" associated with it to indicate that the message "*student_id*" is provided with information about a student identification. The defined type is included at the beginning of the specification. Local variables "*stimulus*?" and "*response*!", both of type "*Stimuli_Or_Responses*", are declared in the schemas corresponding to either an individual class of stimuli or responses (see Figures 7.44 and 7.45). The local variable "*stimulus*?" has "?" as its last character to indicate that it is used as an input. Conversely, the character "!" indicates that the variable "*response*!" is an output.

The "Conditions and Modes" section is then specified; this section defines conditions that are either true or false depending on the system state. The five conditions listed in Figure 7.46 correspond to the threads-based conditions B(a), B(b), D(b), E(b), and A respectively (Figures 7.47, 7.48, 7.49 and 7.50). The conditions are isolated and specified separately so that they can be used in various combinations in the requirements. The third and fourth conditions in Figure 7.46 are set to be equivalent to the predefined conditions described in the "Dependencies" section.

The "Actions" section describe changes in the dynamic state of the system. Actions

Stimuli

$VIT\_input\_student\_identification$

$Telereg\_State$

$id : SId$

$input? : Stimuli\_Or\_Responses$

$input? = student\_id(id)$

Figure 7.44: Z specification, Stimulus section

are described by "commit"s in the threads-based specification. The two actions described in Figure 7.52 correspond to actions described in requirements B or D and F in Figures 7.47, 7.49 and 7.51.

Each of the Z requirements in Figures 7.53 to 7.57 correspond to the threads-based requirements in Figures 7.47 to 7.51. The requirements for the thread consist of the combination of all the requirements described and is specified in Figure 7.58.

7.4 Strengths and Weaknesses of Z

This section presents an evaluation of Z.

7.4.1 Strengths

1. **Z has less potential for ambiguity than semi-formal or informal approaches to specification.**

Responses

```
┌─ VIT_student_identification_acceptance ───────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = last_name_msg
└───────────────────────────────────────────────────

┌─ VIT_invalid_identification_rejection ────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = invalid_id_msg
└───────────────────────────────────────────────────

┌─ VIT_identification_request ──────────────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = request_id_msg
└───────────────────────────────────────────────────

┌─ VIT_connection_rejection ────────────────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = bye_msg
└───────────────────────────────────────────────────

┌─ VIT_hold_or_block_rejection ─────────────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = hold_or_block_msg
└───────────────────────────────────────────────────

┌─ VIT_outstanding_fee_rejection ───────────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = fee_kickout_msg
└───────────────────────────────────────────────────
```

Figure 7.45: Z specification, Response section

Conditions and Modes

The student identification can be found in the student records.

$$\begin{array}{|l}
\hline
\textit{VIT\_id\_found\_condition} \\
\textit{Telereg\_State} \\
id : SId \\
\hline
id \in Students \\
\hline
\end{array}$$

It is the student's third attempt at inputting her student identification.

$$\begin{array}{|l}
\hline
\textit{VIT\_max\_attempt\_exceeded\_condition} \\
\textit{Telereg\_State} \\
\textit{id\_count} : \mathbb{Z} \\
\hline
\textit{id\_count} \geq \textit{MAX\_ID\_COUNT} \\
\hline
\end{array}$$

There are financial holds or disciplinary blocks on the student's record.

$$\textit{VIT\_withhold\_condition} \mathrel{\widehat{=}} \textit{WSRC\_holds\_or\_blocks\_condition}$$

The student has an outstanding fee balance.

$$\textit{VIT\_outstanding\_fee\_condition} \mathrel{\widehat{=}} \textit{COFAC\_outstanding\_fee\_condition}$$

The system is in "Waiting_for_identification_input" state.

$$\begin{array}{l}
\textit{VIT\_in\_identification\_input\_state} \mathrel{\widehat{=}} \\
\qquad [\textit{Telereg\_State} \mid state = \textit{Waiting\_for\_identification\_input}]
\end{array}$$

Figure 7.46: Z specification, Conditions and Modes section

REQUIREMENTS:

Ⓐ If the system is in the "Waiting_for_identification_input" state, Telereg will check whether the [input student identification] can be found in the <student records>.

Ⓑ Telereg shall return a [connection] rejection provided all of the following conditions are true:
a. 'The input cannot be found in the <student records>'.
b. 'It is the student's third attempt at entering her [input student identification]'.
Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

Figure 7.47: Threads-based specification, requirements

Ⓒ Telereg shall return an [invalid identification rejection] and an [identification request] provided all of the following conditions are true:
a. 'The input cannot be found in the <student records>'.
b. 'It is not the student's third attempt yet at entering her [input student identification]'.
Telereg will stay in "Waiting_for_identification_input" state.

Figure 7.48: Threads-based specification, requirements (cont.)

D

Telereg shall return a [hold or block rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'There are <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

The telephone connection shall be terminated. No further processing will be done. Telereg shall commit the system state to "IDLE" state.

Figure 7.49: Threads-based specification, requirements (cont.)

E

Telereg shall return an [outstanding fee rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'The student has an <outstanding fee balance>' as specified in the \Check Outstanding Fee Assessment Condition\.

Telereg will then only allow the student to ask for transactions such as current fee assessment and inquiry on section openings.

Figure 7.50: Threads-based specification, requirements (cont.)

F

Telereg shall return a [student identification acceptance] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'There are no <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

Telereg shall commit to the "Waiting_for_session_input" state.

Figure 7.51: Threads-based specification, requirements (cont.)

Actions

Telereg shall commit to "IDLE" state.

$$\begin{array}{l} VIT\_move\_to\_idle\_state \mathrel{\widehat=} \\ \quad [\Delta\, Telereg\_State \mid state' = IDLE] \end{array}$$

Telereg shall commit to "Waiting_for_session_input" state.

$$\begin{array}{l} VIT\_move\_to\_session\_input\_state \mathrel{\widehat=} \\ \quad [\Delta\, Telereg\_State \mid state' = Waiting\_for\_session\_input] \end{array}$$

Figure 7.52: Z specification, Actions section

Requirements

In the "Waiting_for_identification_input" state, Telereg shall terminate the telephone connection if the student identification cannot be found in the student records and it is the student's third attempt at inputting her student identification. Telereg shall commit to the "IDLE" state.

$$\begin{array}{l} VIT\_Requirement1 \mathrel{\widehat=} \\ \quad VIT\_input\_student\_identification \land \\ \quad VIT\_in\_identification\_input\_state \land \\ \quad \neg\ VIT\_id\_found\_condition \land \\ \quad VIT\_max\_attempt\_exceeded\_condition \Rightarrow \\ \quad VIT\_connection\_rejection \land \\ \quad VIT\_move\_to\_idle\_state \end{array}$$

Figure 7.53: Z specification, Requirements section

In the "Waiting_for_identification_input" state, Telereg shall inform the student that the student identification is invalid and ask for another identification if the student identification cannot be found in the student records and it is not the student's third attempt at inputting her student identification.

$$
\begin{array}{l}
VIT\_Requirement2 \mathrel{\widehat{=}} \\
\quad VIT\_input\_student\_identification \land \\
\quad VIT\_in\_identification\_input\_state \land \\
\quad \neg\ VIT\_id\_found\_condition \land \\
\quad \neg\ VIT\_max\_attempt\_exceeded\_condition \Rightarrow \\
\quad VIT\_invalid\_identification\_rejection \land \\
\quad VIT\_identification\_request
\end{array}
$$

Figure 7.54: Z specification, Requirements section (cont.)

In the "Waiting_for_identification_input" state, Telereg shall terminate the telephone connection after informing the student that she has a financial hold or disciplinary block on her record if her student identification can be found in the student records but there is also a hold or block on the student's record. Telereg shall commit to the "IDLE" state.

$$
\begin{array}{l}
VIT\_Requirement3 \mathrel{\widehat{=}} \\
\quad VIT\_input\_student\_identification \land \\
\quad VIT\_in\_identification\_input\_state \land \\
\quad VIT\_id\_found\_condition \land \\
\quad VIT\_withhold\_condition \Rightarrow \\
\quad VIT\_hold\_or\_block\_rejection \land \\
\quad VIT\_move\_to\_idle\_state
\end{array}
$$

Figure 7.55: Z specification, Requirements section (cont.)

In the "Waiting_for_identification_input" state, after Telereg finds the student identification in the student records, it shall inform the student that she has an outstanding fee balance if she has one.

$$
\begin{array}{l}
\mathit{VIT\_Requirement4} \mathrel{\widehat=} \\
\quad \mathit{VIT\_input\_student\_identification} \land \\
\quad \mathit{VIT\_in\_identification\_input\_state} \land \\
\quad \mathit{VIT\_id\_found\_condition} \land \\
\quad \mathit{VIT\_outstanding\_fee\_condition} \Rightarrow \\
\quad \mathit{VIT\_outstanding\_fee\_rejection}
\end{array}
$$

Figure 7.56: Z specification, Requirements section (cont.)

In the "Waiting_for_identification_input" state, Telereg shall spell out the student's last name after the student identification can be found in the student records and the student is not to be withheld because of financial holds or disciplinary blocks. Telereg shall commit to the "Waiting_for_session_input"state.

$$
\begin{array}{l}
\mathit{VIT\_Requirement5} \mathrel{\widehat=} \\
\quad \mathit{VIT\_input\_student\_identification} \land \\
\quad \mathit{VIT\_in\_identification\_input\_state} \land \\
\quad \mathit{VIT\_id\_found\_condition} \land \\
\quad \neg\ \mathit{VIT\_withhold\_condition} \Rightarrow \\
\quad \mathit{VIT\_student\_identification\_acceptance} \land \\
\quad \mathit{VIT\_move\_to\_session\_input\_state}
\end{array}
$$

Figure 7.57: Z specification, Requirements section (cont.)

The specification of this thread is the conjunction of all of the above requirements:

$$\begin{aligned}&\mathit{Validate\_Identification\_Thread} \mathrel{\widehat=}\\ &\quad \mathit{VIT\_Requirement1} \land\\ &\quad \mathit{VIT\_Requirement2} \land\\ &\quad \mathit{VIT\_Requirement3} \land\\ &\quad \mathit{VIT\_Requirement4} \land\\ &\quad \mathit{VIT\_Requirement5}\end{aligned}$$

Figure 7.58: Z specification, Requirements section (cont.)

Formal specifications allow us to write down user requirements very precisely because precise semantics is associated with the notational symbols used in Z. Therefore, the requirements must first be properly understood before they can be specified. Since the requirements are specified precisely in mathematical statements, they decrease the potential for ambiguity. When a Z specification is derived from a threads-based specification, the codification process can be rather systematic since both approaches have similar structures and specification styles.

2. **Z facilitates requirements validation.**

   Writing a formal specification enables us to find loopholes and inconsistencies in the requirements. Formal specifications are effective at validating requirements because they call for the requirements to be written precisely which implies that the precise meaning of requirements must be understood. Ambiguous natural language requirements have to be clarified before they can be codified. The type checking support offered by Fuzz also ensures the consistency of a Z specification.

3. **Z encourages a rigorous understanding of the requirements.**

   A first step in writing a Z specification is to determine the basic data objects in the application domain. A formal specification effort therefore forces the specifier to understand the fundamental concepts of the system. Besides, a formal specification explicitly defines system invariants which are not specified in natural language specification. While invariants are not necessary for the implementation of the system, they are useful for understanding it.

4. **Z is well-documented.**

   Z is a well-documented methodology. There are several well-written books as well as extensive technical reports on Z. Also, the newsgroup "comp.specification.z" on the Internet is dedicated to answering questions on the use of Z.

5. **A Z specification can aid in testing.**

   Requirements in Z are written in terms of what is externally visible. This is advantageous since it discourages the specification of requirements in terms of internal processing. It is beneficial to independent testing since the requirements are in a testable format.

6. **A Z specification facilitates analysis.**

   Since a specification in Z is machine-readable, it can be used as an input to some form of analysis, be it formal validation, formal verification, or for the generation of test cases during system integration testing. Some analysis, simplified by formal techniques, helps us to achieve a high degree of confidence that a system conforms to its specifications.

7.4.2 Weaknesses

1. **Z requires a high level of technical skills.**

   The Z approach requires a high level of technical skill. A Z specification is almost impossible for a non-expert to read without training. The Z notation consists of many special symbols and graphical representations that have special meanings attached to them. A possible barrier to reading Z specifications is the need for some mathematical background of set theory and first order logic. This can be a major stumbling block for technology transfer to industries.

2. **Z might not be suitable for systems whose requirements are not well-defined.**

   Most likely, if Z were used as the notational language to specify the requirements of large systems, the team of specifiers will not all consist of experts in Z. This can pose a problem if the customers are still trying to define their products and the requirements of the systems are not well understood because specifications might have to be reworked from scratch many times. Hence, Z might not be ideally suited for systems whose requirements have not been well defined. Small systems with simple domain might be easier to specify in a notation like Z. Larger systems with complex domain may not be easily specified in Z.

3. **Z requires special formatting environments.**

   Z symbols are formatted in special formatting environments, for example LaTeX, used primarily in academic settings. Since the symbols are not ASCII-based symbols, they might not be practical in industrial settings where there is no tool support for LaTeX.

4. **Z lacks robust industrial scale tool support.**

   While there is some tool support for Z because of the formality it affords, there is a shortage of robust, industrial scale support tools for analysing Z specifications. However, there has been some work done in academia on a tool called "ProofPower"

[18] which embeds Z in a robust tool called Higher Order Logic (HOL). At the moment, there are not any tools to support reasoning about Z specifications.

5. **Z has some subtleties and possible ambiguities in its semantics.**

   Although Z offers a formal approach to specification, there are some subtleties and possible ambiguities in its otherwise well-defined semantics. This can present some difficulties in that it becomes difficult to build tools to perform analysis on Z specifications beyond type-checking. It is possible to write mathematical statements in Z that are correctly type checked but have no meaning.

6. **Z is not suited to specify finite state machine behaviour as well as concurrency.**

7.5 Conclusion

This section presents some final remarks on Z and on formal methods in general.

7.5.1 Remarks About Z

The purpose of Z is to enhance human understandability of the system and to allow the possibility of formal reasoning and development. Potter *et al.* [24] suggest using Structured Analysis [10] and Z together by capturing the structure of the system with the data flow diagrams and expressing the functional behaviour in Z. Alternatively, they suggest using Z to analyse and understand the requirements before the structure of the system is modelled. Versions of object-oriented Z are also available [6] [28] [13].

7.5.2 Formal Methods

Formal techniques including specification techniques give a high degree of confidence that a system conforms to its specification because the specification is expected to describe the requirements precisely. Other specification languages include the Vienna Development Method (VDM), Communicating Sequential Processes (CSP) and Calculus of Communicating Systems (CCS). While the application of formal methods can be costly, the investment can pay off if formal specifications are re-used for future projects.

The use of mathematics is the key to the usefulness of formal methods. Mathematics provides the ability to reason and it is very precise because its concepts have well defined syntax and semantics. Thus, there can be no ambiguity about what is meant by a mathematical expression. Mathematics is also concise since special symbols are used to denote what would be expressed in several words in natural language. Also, mathematics is stable and will not change with innovations in technology.

Formal specifications are often not only end products of the requirements analysis phase but may potentially be used as input to some form of analysis [19]. The analysis may be of the form of formal validation, formal verification, and system testing. Formal validation enables the clarification of the requirements by establishing the fundamental concepts of the system and the system invariants as well as removing internal processing from the specification. Formal verification involves the use of an interactive theorem prover which takes a formal specification as input to derive a specification of the design or the implementation such that it satisfies the requirements specification. In system testing, a computer-based tool takes the formal specification as a test script and automatically generates test cases, thereby considerably reducing the human effort required to do the task manually.

In most cases, a formal specification is probably not readable and understandable to the customer. This is a major problem since the specification document is seen as a contractual agreement and is used often as a reference document for the customer. The amount of technical skills required to use formal methods is the major stumbling block for technology transfer from academia to industry.

Chapter 8

Discussion

8.1 Introduction

In this chapter, we discuss the relative advantages and disadvantages of the four specification approaches presented in the previous four chapters. Since the main deliverable of requirements analysis and specification is the specification document, the merits of the approaches are judged relative to whether they contribute to producing a useful specification document.

Section 8.2 makes some general observations about requirements analysis and specification. In Section 8.3, the four specification techniques are compared based on their merits to produce a good specification document. Section 8.4 discusses object-oriented techniques and how they compare to Structured Analysis, the threads-based technique, and formal notation Z. Section 8.5 contrasts Structured Analysis with the threads-based technique. The current industrial practice of specification techniques is presented in Section 8.6. Section 8.7 presents the predictions of some industrial practitioners on the kinds of specification techniques which will be used in the future. Section 8.8 reports some of the pressing needs in requirements specification. Section 8.9 reports some of the issues to consider if the techniques were to be used to specify systems that are more complex than the SIS-Telereg system. Finally, Section 8.10 concludes with some final thoughts on the techniques.

8.2 General Observations

It became obvious during the tasks of specifying the requirements of the SIS-Telereg system using the various techniques that it is not enough to count on the professionalism of engineers, even if they have the best of intentions, to write requirements carefully and unambiguously because human nature is prone to error. We have to rely on the technique to force the specifier to write good requirements. In other words, as much as possible the technique must inherently require that the requirements be written in a precise and unambiguous manner. Then, the chances that the specification document will be precise and unambiguous are greatly improved.

But it is not sufficient to have a technique that encourages the specification of precise and unambiguous requirements. For many specifiers, a major stumbling block is how to use the technique to derive the specification document. Ideally, what is helpful is that the approach provides a template for the specification document. Then, the requirements particular to the application can be specified according to the form needed. Alternatively, an approach may describe the step-by-step procedure to derive a specification. However, not all techniques describe how to develop specifications using their notational tools; many techniques only define the notations and describe how the latter must be used. For such techniques, it is important to establish a "process" which will define guidelines to assist in the systematic derivation of a specification document. Other techniques provide some guidelines on how to develop specification documents or specification models and therefore, "processes" for these techniques need not be defined as elaborately as they would for those techniques that do not describe how to develop specifications.

8.3 Comparison of the Four Specification Approaches

The various specification approaches are compared based on their likelihood to produce a specification document which is useful to customers, design engineers, test engineers, and maintenance personnel. In other words, the specifications produced by each approach are judged according to the qualities that a useful specification document must have. Therefore, it is important to consider the readability of the various notations to a wide range of users. Equally important are the amount of technical skill required to use the technique, the maintainability of the specification, the potential for ambiguity, tool support for the technique among others.

Tables 8.1 and 8.2 summarise the relative merits of each of the four specification approaches to produce a useful Requirements Specification. These tables capture in essence the narrative comparisons of the techniques in Sections 8.3.1 to 8.3.14. A point system is used to rate the approaches relative to one another. The point system ranges from 1 (worst) to 4 (best) whereby a score of 4 for an approach indicates that the approach best addresses the criteria of merit compared to the other three approaches. An approach is better than another approach for a particular basis of comparison if the former scores a higher point than the latter. Two approaches scoring the same number of points for a given criteria of merit implies that none clearly addresses the criteria better than the other. Even though an approach may score the same number of points for two different bases of comparison, it does not imply that the approach performs equally well in both areas of comparison. For each basis of comparison, the approaches are rated relative to one another.

While the actual values of the total scores for each approach, as indicated at the bottom right end of the table in Table 8.2, do not have any experimental significance, they give us some intuitive ideas on how the approaches rated comparatively. Overall,

	Produces Readable Specification	Requires Least Amount of Technical Skill	Facilitates Specification Process	Produces Testable Specification
Ad hoc approach	4	4	1	1
Structured Analysis	3	2	2	2
Threads-Based	4	3	4	4
Z	1	1	3	4

	Produces Specification Decoupled From Design	Has least Potential For Ambiguity	Helps to Know When To Stop Analysis	Has Tool Support
Ad hoc approach	1	1	1	1
Structured Analysis	2	3	2	2
Threads-Based	3	2	3	3
Z	4	4	3	3

Table 8.1: Summary of comparison, part 1 (Scale: 1 worst – 4 best)

	Produces Maintainable Specification	Encourages Rigorous Understanding of Requirements	Helps Check Internal Data Consistency	Helps Validate Requirements
Ad hoc approach	1	1	1	1
Structured Analysis	2	3	3	2
Threads-Based	4	3	4	3
Z	4	3	2	3

	Subject To Automatic Analysis	Helps To Know Whether All Requirements Are Captured	*Total Scores For All Criteria*
Ad hoc approach	1	1	20
Structured Analysis	2	2	32
Threads-Based	3	3	46
Z	4	3	42

Table 8.2: Summary of comparison, part 2 (Scale: 1 worst – 4 best)

the *ad hoc* approach is the least effective technique to produce a good Requirements Specification. Of the four specification approaches, both the threads-based technique and the Z approach seem to offer the best alternatives, with the document readability and amount of technical skills required being the two main bases of comparison where the two approaches obviously differ. Structured Analysis is preferred to the *ad hoc* approach.

8.3.1 Readability

The *ad hoc* approach is very accessible because it uses natural language as the medium of communication.

Structured Analysis uses graphical notations which offer better readability than if the requirements were specified textually. However, semantics are associated with the graphical notations and they must be somewhat understood even if only intuitively.

Similarly for the threads-based technique, the use of natural language makes its specification readable. Even though the technique imposes the use of conventions and key phrases on how the requirements are to be written, these conventions are well described and defined and do not reduce the readability of the specification.

On the other hand, Z specifications can be quite hard to read because they require expertise in set theory and predicate logic. It is highly unlikely that all the parties involved in a project, namely the contractors, customers, design engineers, test engineers, and maintenance personnel, would be familiar with a formal notation. Although natural language text accompany the mathematical expressions in order to improve their readability, it is nonetheless the formal statements that must truly be considered by all the users of the specification document.

8.3.2 Amount of Technical Skill Required

The *ad hoc* approach requires the least amount of technical skill compared to the other three techniques. It is the simplest approach to use because it only calls for an understanding of the content and scope of a requirements document which are described in most software engineering textbooks and standards on requirements analysis.

Structured Analysis requires more technical skill than both the informal approaches. The methodology has a graphical notation with a well-defined syntax and a loosely-defined semantics. Therefore, in order to use the methodology, its notational syntax, semantics, intrinsic properties, and conventions must be mastered. Also, knowledge on how to use tool support for the methodology must be acquired. Since Structured Analysis is a relatively mature methodology, it is well documented in textbooks and other kinds of literature.

The threads-based technique requires slightly more technical skill than the *ad hoc* approach because it imposes certain restrictions on the writing style and structure of the specification. These intrinsic properties and conventions of the technique add some learning overhead to the technique despite its use of natural language as the notational tool. The threads-based technique is not widely documented.

The Z formal notation requires by far the largest amount of technical skill because it needs some mathematical background in set theory, predicate logic and type theory. Formal specifications are hard to read without any training. In addition, some expertise is needed to encode Z specifications in special formatting environments, and knowledge in type theory is necessary for type checking. The formal notation Z is well documented and there are a number of textbooks, research papers, and an Internet newsgroup dedicated to discussions on Z.

8.3.3 Does it Facilitate the Specification Process?

The *ad hoc* approach gives the minimal guidelines on how to derive an *ad hoc* specification; it only describes broadly the form and contents of the overall specification. No guidelines whatsoever are given on the writing style, structure and form of the specification. Thus, such an *ad hoc* approach does not facilitate the systematic derivation of the requirements document.

Structured Analysis defines its notations extensively and gives numerous rules and conventions for the usage of the notations. However, there is only a general description of how to derive a Structured Analysis model. More than these guidelines are needed to obtain a specification model.

The threads-based technique provides a template for the requirements document by defining guidelines as to its form, notational conventions, and writing style in detail. These guidelines are not only geared towards deriving as precise a requirements document in natural language as possible but also to assist in the development of the requirements document as a whole. Of the four techniques studied, the threads-based technique is probably the one technique which gives an almost systematic way of deriving requirements.

Literature on the notation Z broadly describes how to organise Z specifications. Since requirements in Z have to be specified so that they describe the states of the system before and after processing an event, a certain style is imposed on Z specifications. Therefore, if the natural language specification – from which the formal specification will be derived – is in a structured format that closely matches that of the Z style, then the derivation of the Z specification will be more systematic. It has been found in the course of this work and during a research project conducted at HCSD [19] that the threads-based technique greatly facilitates the derivation of Z specifications.

8.3.4 Does the Technique Produce a Testable Specification?

It is most likely that the *ad hoc* approach does not produce a testable specification because there are no guidelines to prevent the specification of requirements in terms of internal processing and using different levels of details throughout the document. This approach does not facilitate testing because of its free style format. However, if the *ad hoc* approach follows military standard DOD-STD-2167A, the suitability of the specification document for testing is greatly improved. Standard DOD-STD-2167A mandates the use of "shall" statements to describe testable requirements. When this is used, parsing tools can aid test engineers by extracting testable requirements. The standard also facilitates testing by making the boundaries between various pieces of software explicit.

In Structured Analysis, most of the specification effort is directed towards building a graphical model of the system. However, these models are not testable; they are usually included in the appendices of specification documents. If the system is a relational database, then the customer may request that the model be verified by inspection. However, in other types of systems, the bottom-most layer of the specification, namely the process specification descriptions layer, is, for the most, part the testable portion of the model. Although the process specifications are written in natural language, they explicitly identify the inputs and outputs of a process which is of some use during testing. Overall, the methodology is not as suited to producing a testable specification as the threads-based approach.

The threads-based technique is designed to produce testable documents. This is facilitated by a black box view of the system. The structure afforded by the threads-based technique facilitates the generation of test cases by computer-based parsing tools. That is why "shall" statements are used to indicate testable requirements, the phrase "as included in" is used to indicate where data comes from, each requirement is written in a separate paragraph, and so on. Finally, the stimulus-response model adopted by the technique makes system testing quite explicit.

The machine-readability of Z specifications enables the computer extraction of test cases from the specifications. Since the statements are mathematical and connected with logical connectives, they can be manipulated with mathematical rules to derive all sets of possible test cases. Further, Z specifications are suitable for testing by specifying requirements in terms of what is externally visible.

8.3.5 Is the Specification Decoupled From Design?

The *ad hoc* approach does not explicitly have guidelines to prevent the specifier from including design issues in the specification.

In Structured Analysis, the requirements are partitioned top-down to present a hierarchical view of the system. This may lead to a tendency to partition more than what is needed and thus stepping into design issues. In other words, the methodology may indirectly conflate design and specification since it provides no guidelines on when to stop partitioning requirements.

The threads-based technique provides for certain mechanisms to encourage the specifier to express requirements in terms of externally visible events or behaviour. This may potentially result in a specification document that does not contain design issues.

Z specifications encourage codification to be moved forward into the analysis phase. In Z, requirements have to be specified such that they describe the system state before and after processing an event; consequently requirements have to be specified from an external point of view. Z naturally discourages description of internal processing.

8.3.6 Potential for Ambiguity

No matter how careful a specifier may be when using the *ad hoc* approach, the potential for ambiguity is great since ambiguity is inherent in natural language. Further, this approach provides no guidelines to minimise the potential for ambiguity.

To the extent that the graphical notations in Structured Analysis have some semantics associated with them, ambiguity is reduced. But these semantics are only loosely defined and are sometimes open to interpretation. So, there is still room for ambiguity. Also, even though process specifications only make up the bottom-most layer of the specification, they are specified in natural language. Nevertheless, Structured Analysis is potentially less ambiguous than the *ad hoc* natural language approach.

The use of natural language in the threads-based technique can potentially result in ambiguity. However, the technique provides for various rules to minimise the effects of the ambiguous nature of natural language. For instance, the technique advocates the use of enumerated lists for conditions and actions, limited vocabulary, special delimiters for data entities, use of special key phrases to make requirements clearer, and elimination of ambiguous phrases such as "and/or" and "it". Therefore, the threads-based technique is potentially less ambiguous than the *ad hoc* approach.

Formal notations force the specifier to express requirements precisely because well-defined semantics are associated with the notations. Therefore, formal specifications are less liable to be misunderstood. On a relative scale, formal techniques have the least potential for ambiguity compared to the other three approaches.

8.3.7 Does the Technique Help in Knowing When to Stop Analysis?

In the *ad hoc* approach, there are no quantitative means to assess whether all the requirements have been addressed. It is not enough to check that at least some requirement has been specified for each section of requirements.

In Structured Analysis, considering all the inputs and outputs of the system and ensuring that they are somehow processed can give some indication of how much more analysis is needed. Tool support for balancing a model of the system can assist in checking that all inputs and outputs are processed.

In the threads-based technique, checking that all external events have been taken into account is a quantitative factor to help decide whether analysis is complete. Moreover, the technique requires that any necessary validation of inputs be described explicitly before the requirements for rejection and acceptance of inputs under various combinations of conditions are specified in separate paragraphs. These structural impositions on the specification force the specifier to consider all stimulus-response combinations, thereby encouraging the chances that most requirements are specified.

Z specifications express the requirements of a system by describing the state of the system before and after the occurrence of an event. A possible indication that analysis is complete is checking that all events are examined. If the overall structure and organisation of Z specifications proposed by textbooks are followed, then they can ensure that most aspects of the system are considered and specified.

8.3.8 Tool Support for the Technique

A word processor is the only tool support needed by the *ad hoc* approach. Because of the lack of structure in the document, the success rates of any type of natural language understanding tools would be quite low.

There are various commercial CASE tools that support Structured Analysis. The facilities offered by the CASE tools may range from checking the syntax of the notation to data consistency in the data dictionary and balancing of the model. However, at present there are no tools that support very large systems. For instance, the system developed by HCSD is so complex that it exceeded the size of the data repository of their tool support and several separate databases had to be developed. This defeats the point of using Structured Analysis which offers the advantage of balancing of the model [21].

Since the threads-based technique is not well known, there are no commercial tools for it. However, it should be possible to build some type of parser to extract information from the specification because of the structure in the document and the use of keywords and delimiters.

There is no robust industrial scale tool support for Z. There are a few commercially and academically developed tools for type checking and for some analysis on the specification. Since Z specifications have to be written in an encoded form because they contain non-ASCII symbols, they need special formatting environments which are used mostly in academic circles.

8.3.9 Maintainability

The free format used in the *ad hoc* approach causes its specification to be hard to maintain. If some functionality of the system is removed, revisions to the specification may not be localised. The whole specification would have to be checked to ensure that all references to the requirements throughout the document are removed. This can be labour intensive if the system is a complex one and the specification document is large. Further, since the approach does not have any mechanisms to discourage specifying design details, then if decisions are made during the design phase which conflict with those suggested in the specification document, then the latter must be changed to reflect the changes. There is, however, tool support to identify, control, track, and report changes made to the specification.

In Structured Analysis, if sections of the specification are removed, the resulting model may not be consistent anymore and may need restructuring. Depending on the changes, the model may not be easily modifiable and maintainable. Moreover, since the methodology can indirectly contain design considerations through the partitioning of the system, if the design is changed, any design choices made later on may have to be reflected in the specification model. This methodology is best suited for systems whose requirements are frozen and do not change continually [22]. Changes to the functionality may unbalance the model requiring it to be restructured. Although some commercial tools may provide version control and some consistency checking, many CASE tools do not provide mechanisms for identifying and reporting changes made to the model in the same way that some tools may provide strikeouts and red lines to show changes made to textual descriptions.

In the threads-based approach, most aspects of the functionality of the system are associated with some set of stimuli. If some functionality of the system is to be removed, it can be traced back to the specification unit that describes the processing incurred upon the receipt of the stimuli, and changes can easily be made. The mapping between stimulus and functionalities facilitates maintenance. Tool support is available to track and report changes made to the specification.

Depending on how well abstraction is used to organise the specification of requirements in Z, changes to the functionality of the system may be made easily. Ideally, a Z specification naturally avoids redundancy so that revisions will be limited to a few obvious locations. However, there is no tool support to identify and control changes made to the specification.

8.3.10 Does the Technique Encourage a Rigorous Understanding of the Requirements?

In the *ad hoc* approach, there are no mechanisms to ensure that all the requirements have been covered. So, specification writers can easily miss important aspects of the system.

In Structured Analysis, the identification of all the external inputs and outputs of the system reduces the likelihood of missing some functionality of the system. However, while the principle of balancing can ensure that the data inputs and outputs are somehow used, there is no guarantee that the processing of the data is specified correctly. The structural decomposition of the system in a top-down fashion may assist in the understanding of the system.

On the other hand, the threads-based technique requires that requirements be written in a certain pattern, namely the specification of the validation of inputs followed by the requirements for all possible acceptance and rejection responses of the inputs in separate paragraphs. Therefore, the various possibilities for an input are investigated in a systematic manner and this reduces the chances that a requirement is overlooked. With the technique expressing the actions on entities in the system upon the occurrence of external stimuli, considering all the external stimuli and responses can give an indication that most of the requirements are taken care of, consequently implying that the system has been rigorously analysed.

To derive a Z specification, it is important to first identify the basic entities in the system and to understand the fundamental concepts of the system so that the rest of the specification effort can build on those concepts. So, a good understanding of the system as a whole is necessary before the specification of the system can be derived. Z encourages the specification of invariants in the system, which can only be done if there is a rigorous global understanding of the system.

8.3.11 Internal Data Consistency of the Specification

In the *ad hoc* approach, manual reviews are required to check the internal data consistency of the document. Even then, chances of overlooking inconsistencies are great. This manual process can be time consuming, especially if the system is complex.

To the extent that Structured Analysis permits automatic balancing of the specification with some CASE tool support, the overall data consistency of the model and the accompanying data dictionary can be checked. However, there is no automatic way of checking that an input to a process is actually used; this is especially problematic in large complex systems where inconsistencies may be overlooked by manual checking.

The threads-based technique requires that data dictionary entries be enclosed by special symbols and that local names used in the specification units for stimuli and responses be delimited as well. This makes it easier to locate references to data. Further, this facilitates the extraction of references of data entities from the specification by computer-based tools. Moreover, the phrase "as included in" is used in the technique to indicate where a piece of data comes from, thus helping in checking data consistency in the document. However, checking whether a piece of data is really used must be done manually.

In Z specifications, a type checker only ensures that data is properly typed. There are no other mechanisms to guarantee or check data consistency.

8.3.12 Does the Technique Informally or Formally Help to Validate the Requirements?

Requirements in the *ad hoc* approach are specified in a narrative and textual style which makes it hard to assess whether all the requirements are specified and that there are no inconsistencies. Further, we are so used to reading natural language in everyday life that we may fail to discover loopholes or ambiguities because we may intuitively add our own interpretation to what we read.

Structured Analysis is not designed to help validate requirements. However, the use of graphics together with the presentation of the requirements in a hierarchical manner may facilitate communication during reviews with the customer, consequently helping to discover inconsistencies earlier in the software development cycle.

In the threads-based technique, since each requirement must be written in a separate paragraph that must be understood in isolation, all possible stimulus-response combinations pertaining to a specific requirement must be explicitly described. This order and pattern imposed on the structure of the specification by the approach forces the specifier to look at these combinations for the requirements which increases the chances of finding loopholes in the specification.

At the very least, the expected content and outline of a Z specification can ensure that both normal and error condition requirements will be specified. Moreover, since a Z specification is machine readable, it can be used as input to some form of formal validation.

8.3.13 Does the Specification Technique Facilitate Automatic Analysis?

Any form of automatic analysis on an *ad hoc* specification can only be performed by natural language understanding tools. However, the lack of structure in the document greatly decreases the chances of success of any meaningful analysis that may be attempted.

No other type of analysis may be done on a Structured Analysis model except for those offered by the tool support because it is mostly graphical and its semantics are not well-defined. Since natural language is used in the process specifications, any kind of analysis on them would be limited unless some structure were added to these textual descriptions.

Intelligent analysis can be done rather easily on a threads-based specification because of the presence of structure, keywords, and delimiters. The presence of these notational conventions greatly facilitates the task of the parser.

Because Z specifications are machine readable, they are most likely to be amenable to all kinds of automatic analysis.

8.3.14 Have All the Requirements Been Captured?

The lack of structure and guidelines in the *ad hoc* approach makes it hard to assess whether the specification is complete. However, the prototype outline of an *ad hoc* specification ensures that at least some requirements are specified for each type of requirement.

In Structured Analysis, the identification of all inputs and outputs and their processing may help evaluate whether the specification is complete. The partitioning of the system into a context diagram, several levels of data flow diagrams and a final level of process specifications may indicate when the specification is complete.

The threads-based technique encourages the specifier to consider all possible stimulus-response combinations for each pair of stimulus-responses. This can help ensure that most requirements are covered when all external stimuli and responses are determined.

Literature on Z gives outlines on the form and contents of Z specifications. These can aid in ensuring that all the various components of a Z specification are presented. Identifying all the events of the systems can also assist in determining the completeness of the document.

8.4 Comparison With Object-Oriented Techniques

To present a complete picture of the scope of specification techniques, it is important to consider object-oriented techniques as well. Object-oriented techniques were not as closely examined as the other four techniques described in this thesis. Both Structured

Analysis and object-oriented techniques use "similar modelling constructs" [27]. For example, both techniques use data flow diagrams.

8.4.1 Object-Oriented Techniques

Object-oriented techniques are increasingly popular. The term "object-oriented" refers to the organisation of software as a "collection of discrete objects that incorporate both data structure and behavior" [27]. Object-oriented techniques are claimed to promote evolutionary software development and reuse because the functional requirements of the system are more likely to change than the application domain.

Since object-oriented techniques are relatively new and thus immature, it is not apparent at the moment which specification technique(s) best support(s) object-oriented design. There are many differing views about how requirements must be written to cleanly map into the object-oriented approach. Research is actively being done in academia and industry.

Many people often mistake the use of languages such as C++ as being synonymous to the use of object-oriented techniques. Increasing tool support for software libraries in object-oriented programming languages sway many organisations to use object-oriented programming – they may not be properly using the features that make object-oriented languages powerful and useful, namely inheritance and polymorphism. Many industries may also be using object-oriented programming after they have specified their requirements in a specification technique that does not lend itself to an object-oriented approach.

Object-Oriented Analysis (OOA) offers an information model of a system. In complex systems, however, it might be necessary to model the process and control aspects of the system in addition to the data model. Rumbaugh *et al.* [27] advocate an object-oriented analysis methodology which presents three views of the system: data, control, and function. However, attempts to depict all three viewpoints of a system can be quite time consuming and complex unless the task is assisted by sophisticated CASE tools which are currently unavailable for large systems [5].

There are few guidelines on how to write requirements specification using OOA. However, other techniques such as the threads-based approach can be used to derive the functional decomposition of the system around the basic objects of the system and by identifying real domain inputs and outputs. Then, this preliminary specification can be used to derive the OOA specification since there is a correlation between the system objects in both techniques.

8.4.2 Structured Analysis vs. Object-Oriented Analysis

Structured Analysis partitions a system based on common functionality. In Structured Analysis, two views of the system are presented: control and function. Yet, an OOA model based on Rumbaugh *et al.* is not equivalent to a Structured Analysis model together with a data model of the system because the two methodologies are fundamentally different. Intuitively, the object-oriented approach deals with "ask not what the system does but to what does it do it" [5]. Models obtained from the two techniques would be different even though they both represent the control and functional behaviour of systems. Some of the notational aspects of Structured Analysis can be used to support object-oriented analysis e.g. Data Flow Diagrams, Data Dictionary entries, State Transition Diagrams.

Although Structured Analysis is primarily based on functional decomposition, it also involves the organisation and decomposition of data. This organisation and decomposition of data may not be compatible with the organisation and decomposition of data in an object-oriented approach. However, if a Structured Analysis model is structured such that the real data entities in the system are identified as the data flows in the model, then the Structured Analysis model can be used as a preliminary specification document to derive an OOA specification.

8.4.3 Threads-Based Technique vs. Object-Oriented Analysis

The threads-based technique is not an object-oriented approach to specification. In fact, the threads-based technique gives a functional decomposition of the system and produces a document which enables the generation of test cases with minimal difficulty. The technique describes the effects of external stimuli on entities of the system.

However, the technique lends itself to OOA because it discourages the introduction of design structure in the requirements. Also, the technique calls for the identification of actual system inputs and outputs which is useful for object-oriented analysis and design.

8.4.4 Z vs. Object-Oriented Analysis

While formal specification using the language Z is not an object-oriented approach, it has an object-oriented flavour. The conventional uses of standard Z as described in Potter *et al.* [24] exhibit features of object-oriented approaches: information and functionality are encapsulated in constructs called schemas, abstraction is enabled by the specification of requirements in terms of external processing and by identifying the primitive data objects of the system before specifying what the system does, and polymorphism can be addressed by the use of generic definitions. There is no well known way to achieve the effect of inheritance easily in Z. A Z specification may potentially be used as a preliminary requirements document to derive an OOA specification.

Z has been extended to facilitate specification in an object-oriented style. There are numerous variants of object-oriented Z; Object-Z, Z++, Object-Oriented Z Environment (OOZE) are some of them [13] [28]. For instance, Object-Z introduces the concept of a *class schema* "which captures the object-oriented notion of a class by encapsulating a single state schema with all the operation schemas which may affect its variables" [28].

The class schema also defines a type whose instances are objects. Further, most of the object-oriented variants of Z support mechanisms for inheritance and polymorphism.

It is important to note that, just like it is not necessary to use an object-oriented programming language to implement an object-oriented design, it is also not necessary that an object-oriented variant of Z be used to support an object-oriented method of analysis.

8.5 Structured Analysis vs. Threads-Based Technique

Although both techniques offer a functional decomposition of the system, they are not equivalent. The threads-based technique presents an external stimulus-response model compared to the process model favoured by Structured Analysis. The threads-based technique describes what the system does upon the occurrence of an external event whereas Structured Analysis describes how data is transformed. In Structured Analysis, design structure can be introduced during the partitioning of requirements whereas in the threads-based technique, there are guidelines to discourage the inclusion of design structure in the requirements.

8.6 Current Status of Requirements Specification

At the moment, most companies use the *ad hoc* specification approach without any formalism, with cost and tight schedules being the primary reasons for such a choice. Some of the few companies who do Structured Analysis do not use it well. Structured Analysis is applied in a fairly *ad hoc* manner and practitioners are fairly immature in their use of the methodology. They are often preoccupied with how to apply or use the method instead of concentrating on what they want to model. Many have to tailor the methodology to their needs because of budget and schedule constraints [5]. Techniques similar to the threads-based technique may be used in some companies. At present, the use of formal specification techniques is rather limited because of the probable cost and time needed to use them well. However, they are slowly gaining popularity in industries. Object-oriented techniques are currently being investigated but they are relatively new and immature techniques. Although the use of object-oriented programming is becoming predominant in many companies, this must not be confused to mean that OOA techniques are well-understood and widely used. Industries who use object-oriented programming without using a specification technique that leads to object-oriented design and programming would not gain most of the benefits of object-oriented programming.

While it would seem reasonable to expect that most companies would try to move away from *ad hoc* approaches because of the great, and proven, risks of producing a Requirements Specification which does not fulfill any of its uses, this is not likely to happen in the future. Because companies are primarily motivated by profit, they are likely to invest in techniques which can bring them a profit and which will give them a competitive edge. Further, they are concerned about the availability of tools that are fully implemented [21]. However, many of the techniques only have prototype tool support or tool support at the developmental stage. The lack of an obvious winning specification technique is a deterrent for many companies to invest in better specification techniques than the one they currently use.

8.7 Predictions for the Future

It is expected that more companies will move away from *ad hoc* approaches towards more formal techniques such as Structured Analysis and Z. Other companies will try to be inventive and come up with approaches especially geared for their application problem [21].

It can also be expected that the iterative approach [22] to software development will gain in popularity because more attempts will be made to build systems of greater complexity with requirements that may not be well-defined at the beginning of the project. However, this implies that military standard DOD-STD-2167A will need to be updated because the latter advocates a waterfall model of software development [21].

8.8 Some Pressing Needs in Requirements Specification

The ability to do analysis on the specification using computer-based tools would be highly beneficial. For instance, the automatic generation of test cases directly from the requirements document would be time saving and as well as money saving and would remove the tedium associated with manual derivation of test cases.

Clark [5] reports the need for better CASE tool support that is robust, gives high performance, and addresses concurrency. For instance, for complex systems, it is impossible to really use object-oriented techniques such as the one proposed by Rumbaugh *et al.* [27] because of the lack of adequate CASE tool support. Some so-called object-oriented CASE tools are really derivatives or enhancements of Structured Analysis tools.

Reusing requirements documented for systems derived from the same product line would lead to considerable savings of cost. It is being investigated at the moment whether the use of object-oriented approaches will assist the re-usability of requirements.

8.9 Scaling Up to Large Complex Systems

Obviously, for large complex systems, many engineers will be involved in requirements analysis and specification during the initial phases of the project. Therefore, communication problems are likely to occur which were not encountered to the same degree as during the specification of a small project as the SIS-Telereg system.

The bigger that the problem is, the more reliant the contractor must be on effective communications, problem decomposition strategies, and control metrics to ensure that the problem is manageable, that issues are being resolved and that sufficient progress is being made.

A general precept is that the bigger the team, the more rigorously defined the process for requirements specification.

8.10 Conclusion

Different application problems may call for different specification techniques. To choose a technique, the life cycle of the system and the features of the techniques must be considered to determine what should be modelled. Practitioners must not use a standardised approach independent of the application. However, to opt for the most appropriate technique for a particular application will require a company to invest time and money to learn and train their engineers in the methodology. This might be a big deterrent to using the best technique for an application rather than the one that is best known to the company.

It is to be recognised that the uses and purposes of the specification also drive the selection of an approach. The choice of a requirements specification technique may depend on the type of design that is wanted because the specification must map to the required design. For example, if the customer requests object-oriented design, then using Structured Analysis for requirements analysis may not be appropriate if the model does not identify the real data entities of the system as the data flows in the system.

Another factor influencing the choice of technique relates to what the contractor plans for possible future application of the requirements specification. For instance, if the contractor wants to stay in the same line of product, it would be beneficial for him to customise the requirements documents to the needs of other customers. On the other hand, if the contractor prepares a requirements document which will be used for only one product, then the document may not need to be adaptable.

If the customer does not have a well-defined understanding of the requirements of a proposed system, then it may be necessary to develop early prototypes of the system to assist in their investigation of what is wanted. In this case, an object oriented approach to specifying requirements may be suitable. Alternatively, the contractor may choose a functional approach which provides the flexibility to change requirements without massive restructuring each time the requirements become better understood.

But it must be kept in mind that the technique itself does not guarantee that a specification document will have the attributes of a useful specification document, e.g. completeness, testability, unambiguity, and conciseness. "A method is a necessary condition but not a sufficient condition" [5]. A process employing the technique to its best advantage must be established to ensure that these attributes will be present. Furthermore, the organisation structure must ensure that the process is followed. Clark [5] reports that whether specification guidelines are followed is a function of team leadership, the type of product expected and the previous establishment of an outline of what the specification looks like.

Clark proposes that the kind of end product wanted can provide guidance as to what should appear in each section of the specification document. Then, once a technique is chosen based on what the final specification must look like, the issues of notation to use and corresponding tool support can be addressed. Training of employees may be achieved by a combination of hiring external consultants, providing employees with appropriate literature and hands-on training.

What has been evident throughout the discussion on the relative merits of the various specification approaches is that none of them is ideal in all respects. Choosing the appropriate one for a particular application depends on what the specification will be used for. However, before this choice can be made, the contractor has to be aware of what the advantages, disadvantages, and limitations of the techniques are so that an informed choice can be made. The work for this research has been partly motivated to provide developers with the information that they need to produce better requirements specifications. Clark [5] supports the mastering of a technique on a small project to determine the limitations of the approach before an attempt is made to apply the technique to a larger system.

Chapter 9

Conclusion

9.1 Outline

This chapter presents the conclusions of this work in a non-traditional manner by considering a hypothetical situation where a specification approach has to be chosen to specify a defined system. Section 9.2 gives the scenario for the hypothetical situation. Section 9.3 presents a high level summary of the system to be specified and Section 9.4 details the proposed strategy for the derivation of the Requirements Specification of the system by extracting from the results of this work. Section 9.5 presents a critique of the proposed solution and describes how the conclusions of this work can be generalised to other systems as well.

9.2 Scenario

I am the CEO of the "Wong Company and Associates"(WCA), an up-and-coming system company which was mentioned among the "100 companies most likely to succeed" in a 1994 Fortune Magazine edition. WCA has gained a well-earned reputation of delivering its customer products well within budget and time limits while using good and state-of-the-art software engineering practices.

In an effort to expand its areas of expertise, WCA put a bid to build the BATS[1] system (Baby Air Traffic System) for the island of Mauritius and won. Competing companies

[1] Credit is given to Jeff Joyce for the name and features of this system.

were corporate giants such as Hughes Mauritius, IBM Mauritius, Harrel Mallac, and Blanche Birger. The Mauritian government saw the need to modernise the country's air traffic control system in view of the booming tourist industry in the last five years and its international recognition as a central business trade centre in the Indian Ocean and in African countries. WAC proposes to build the BATS system with a budget of $40 million and a work force of 200 engineers over a four year period.

9.3 The BATS System

The BATS system is primarily a control-oriented system with data processing capabilities and some safety-critical aspects. An understanding of the system relies heavily on domain expertise in air traffic control.

BATS shall provide oceanic traffic control services for the Mauritius Air Traffic Control Region (MATCR). BATS will receive flight plans and flight plan updates from operators at the local control centre in Mahebourg as well as from external agencies of other countries for regions adjacent to MATCR.

A flight plan shall include aircraft identification, aircraft type, pilot's name, departure aerodrome, destination aerodrome and profile. The profile of a flight plan is a sequence of space and time coordinates. Upon receipt of the flight plan, BATS will check that the flight plan is valid and a non-duplicate and that the filed route is conflict-free. BATS will receive primary and secondary radar inputs from its radar facilities located at Le Morne, Flic en Flac, and Curepipe. Primary radar input will show direction and range of radar targets. Secondary radar input will provide identification, attitude and altimeter setting of radar targets. The secondary radar inputs are only available for aircrafts with operational transponders.

BATS will also receive weather information from the Mauritius National Weather

Office and from the national weather offices of Reunion Island, Madagascar, Seychelles, Rodrigues, and South Africa. BATS shall display the profile, position, speed and flight level of every controlled aircraft. The profile will be displayed as a projection onto the surface of the controlled region (i.e., a looking down view). However, a controller may optionally select to display a projection that shows the altitude of the aircraft (i.e., a sideways view). BATS shall also display the position of every object detected by primary radar. BATS shall maintain a profile for each aircraft.

As reports are received by radio communication from each aircraft, operators shall update route points associated with the flight plans of active flights. BATS will automatically update the profile for yet-to-be-reached route points. BATS will automatically predict conflicts based on international standards for aircraft separation and alert operators. The operator will be alerted at least five minutes before the conflict is predicted to occur. The flight plans for aircrafts destined for Mauritius will be automatically sent to the facility responsible for domestic air traffic control in Mauritius. When an aircraft is about to exit MATCR and enter an adjacent region, the flight plan of the aircraft will be automatically sent to the external agencies responsible for controlling the adjacent regions at least ten minutes prior to the aircraft exiting MATCR. Flight plans will be retained for a minimum of one week after the flight has terminated.

9.4 Proposed Solution

As the Project Manager and Technical Director of the company, I am responsible for proposing a suitable specification approach for this project. Various factors must be taken into consideration during the selection of the most appropriate specification approach for the BATS system. The uses and purposes of the Requirements Specification must be considered as well as the type of design envisioned for the system.

9.4.1 What Does WCA Want to Achieve?

As can be expected, it is crucial for WCA that the Requirements Specification be readable and understandable to the Mauritian government and WCA engineers and subcontractors. This is so that our customers can increase their confidence that WCA understands what they would like built and for them to see how WCA is progressing. Equally important is that they are able to read the Requirements Specification and can fully participate and contribute to correcting misunderstandings and omissions in the Requirements Specification during joint reviews. Therefore, the use of unfamiliar mathematical notations may not be appropriate because WCA contacts with the customer will consist mainly of air traffic controllers and government officials.

Further, WCA is considering the possibility of staying in the same line of product for air traffic control systems of surrounding islands and African countries. While the feasibility of such a venture has not been fully determined, WCA is nevertheless interested in producing a Requirements Specification that is easily modifiable to meet the needs of air traffic systems particular to other countries. It is anticipated that the bulk of the requirements will stay the same and that an architecture similar to the one used with the BATS system will be reusable for other air traffic control systems. Hence, using the *ad hoc* natural language approach is totally inappropriate because of the horrendous maintenance problems that will incur with a system of the size of BATS. Also, the *ad hoc* natural language approach does not provide guidelines to encourage the specification of requirements related to a particular concern in one place. Thus, it is hard to localise and update the system requirements to match the needs of new prospective air traffic systems. Both Structured Analysis and the threads-based technique are possible alternatives. However, with Structured Analysis, if substantial changes have to be made in the functionality of the air traffic system, then the model may need substantial restructuring. The threads-based technique is more advantageous than Structured Analysis if major changes are made because in this technique, most aspects of the functionality of the system are usually associated with some set of stimuli. Therefore, if some functionality is to be removed, it can be traced back to the specification unit that describes the processing incurred upon the receipt of the stimuli. The structure imposed by the technique facilitates the localisation of requirements.

WCA is interested in following an object-oriented approach during the design and implementation phases. Thus, it is important that the selected specification approach lends itself to an object-oriented design approach. An object-oriented approach is favoured because it allows a risk driven iterative method which is beneficial in two ways. Such an iterative method can potentially mitigate project risks and can provide the customers increased visibility into progress made in the system development. Further, an object-oriented approach supports evolutionary design by allowing incremental additions to the functionality of objects in the system [22].

9.4.2 How is WCA Going to Achieve its Goals?

Since none of the specification techniques are complete, a combination of techniques is necessary for analysing the requirements and deriving a Requirements Specification that is useful to subsequent stages of system development and is readable and understandable to customers, design and test engineers and maintenance personnel. I am not proposing, by any means, that all the techniques be applied fully to the whole system. Rather, I am proposing to use each technique as it best benefits us and use it only enough for what it best addresses. Therefore, it is proposed that a sequence of techniques be used: as a first step, the technique that best helps understand and uncover the system requirements is used and then, in a second step, the specification of the requirements is done using an approach that is conducive to producing a Requirements Specification that will meet the intended goals of the system. Other techniques may be selectively applied to parts of the system.

More specifically, I propose that Structured Analysis be applied just enough to help us acquire a rigorous understanding of the requirements and to organise the requirements. A full model is not developed because Structured Analysis models are not testable. Also, they are not decoupled from design because design structure is inherently introduced during partitioning. Further, Structured Analysis will not be used to specify the system fully because the model cannot be subject to any form of automatic analysis other than syntactic checking by the CASE tool support.

Structured Analysis is a suitable technique for the preliminary investigation of requirements analysis because the methodology can give us some intuitive idea of how the requirements fit together. The lack of precise semantics gives the flexibility of specifying the requirements without having to worry about details in the notation. Further, the top-down view of the system helps us assimilate the system requirements as it presents a progressively detailed view of the system. Since the BATS system is control-oriented, the real-time extensions of Structured Analysis are well-suited to depict the finite state machine behaviour of the system. The exercise of using Structured Analysis also encourages the derivation of the content and format of the various data entities. It is important that the actual system inputs and outputs be identified as data flows in the model so that they can be used when the requirements of the system are fully specified. Also, the partitioning of requirements afforded by the methodology can help us organise the requirements. Structured Analysis is applied enough that it gives us a preliminary understanding of the requirements so that the allocation of requirements into threads specification units can be done smoothly.

The threads-based technique is proposed as the specification approach to specify the

Requirements Specification for the BATS system. The threads-based technique is the best-suited technique for this application for numerous reasons. The guidelines and conventions imposed by the technique on the structure of the document provide a template for the Requirements Specification and thus facilitate the systematic specification of requirements. The specification process for the project need not be elaborately established because the technique inherently has guidelines to facilitate requirements specification in a systematic manner. The established process may impose conventions that are specific to the air traffic control application domain and to suit the Mauritius Standard on System Development.

Equally important is that the threads-based technique supports the risk driven iterative approach in that it discourages the introduction of design structure in the requirements. Moreover, the specification units in the technique are purposely structured to stand alone and to be well-defined. This enables the selective application of requirements to the design architecture in an iterative manner.

The use of natural language as the notational tool in the threads-based technique contributes to the Requirements Specification's readability and understandability to all parties involved in the BATS project. This also implies that not much technical skill is required for the specification effort and minimal training expenses are incurred. The conventional style imposed by the technique can ensure that a consistently styled document will be derived and this facilitates the tasks of design and test engineers during the later stages of software development. Further, a consistently styled document where requirements are specified in terms of what is externally visible makes independent testing easier. The specification of requirements from an external point of view discourages the introduction of design details in the specification, thereby decreasing the burden of maintenance. The identification of all external stimuli and responses of the system can give some indication whether all the requirements have been captured. The use of special notational conventions for data entities and "shall" statements for testable requirements can facilitate information extraction by computer-based tools and thus assist in testing and analysis of the specification for internal data consistency.

The system's finite state machine behaviour captured during Structured Analysis can be reused. As remarked earlier, it is crucial that the inputs and outputs of the finite state machine behaviour of the system correspond to the actual externally visible inputs and outputs of the system. The high level Structured Analysis model is attached to the threads-based specification to give a graphical overall picture of the functionality of the system and the interrelationships between the various functions. Hence, the Structured Analysis model provides the global mapping of how the specification units in the technique fit together and shows the path between inputs and outputs.

Besides its data processing and control behaviour, the BATS system has requirements pertaining to the prediction of aircraft conflicts. Since these aspects are safety-critical requirements, formal techniques are selectively applied to those parts of the system. The usage of formal methods can increase our confidence in the system. For a systematic specification of requirements in a formal notation, the latter notation must have the same specification style and document structure as the threads-based technique and must also require that requirements be specified in terms of what is externally visible. Formal notations are machine readable and can be subject to automatic analysis. Formal notations are not applied to the whole system because of the lack of manpower proficient in formal techniques and budget and time constraints.

9.5 Critique

The proposed solution for the most appropriate specification approach for the BATS system calls for the usage of three different techniques. It would seem that the solution would be costly and time-consuming since training of all software engineers would be needed. However, this is really not the case. The proposal would actually just call for a small team of people to know Structured Analysis, for all of the software engineers to be familiar with the threads-based technique and for a few experts in formal methods.

While all of the software engineers will be sent to attend courses on air traffic control, only a small team will actually have interaction with the customers and the air traffic controllers. This small team will be the cluster of in-house experts on air traffic control. This team will consist of high level management and technical people experienced in requirements analysis and Structured Analysis and people experienced in air traffic control. The team members who are not familiar with Structured Analysis need not know more than the notation used and only need to have some intuitive ideas of the semantics of the notation. The team will be responsible for the analysis of the system and accumulating domain knowledge on the MATCR. The engineers on this team will use Structured Analysis to obtain the overall picture of the system and to understand how all the requirements fit together. In parallel, the other software engineers will be attending courses on air traffic control and learn to be familiar with the threads-based approach. Since only a preliminary model of the system will be built, a robust CASE tool is not a necessity. It may even be reasonable to do Structured Analysis with pen and paper.

This first step of using Structured Analysis to acquire some rigorous understanding of the requirements is essential. Other techniques are not as conducive as Structured Analysis to help understand the system. For instance, while the threads-based technique may be ideal for the actual specification of requirements, it does not assist in getting a first cut at understanding the requirements and giving a way to start analysing the system. Once the big picture of the system is obtained, the chances that some important concepts are missed and discovered during later stages are decreased.

Once a preliminary understanding of the system is gained, the requirements can be tentatively allocated into the specification units of the threads-based technique in a straight-forward manner. A series of reviews are necessary before the final allocation process is approved by the customer and the specification style is approved by the technical staff. Given the large size of the system, almost all software engineers will be involved in this task of specification. Since the threads-based technique is used, the engineers will only be required to follow the guidelines and conventions of the technique closely. The small team of in-house experts in the application domain will assist the specifiers. The threads-based technique is also appropriate for the BATS system because there are many input sources in the system and the stimulus-response model adopted by the technique suitably addresses these issues. Real-time requirements are also adequately addressed by the technique.

The safety-critical aspects of the system will be attended by a specialised team of formal methods experts. External experts may be brought in and experienced in-house engineers may be put on that team. While some expenses will be incurred upfront to bring in experts, it is arguable that it is a worthwhile investment since it can potentially contribute to increasing the reliability of the system. While, at this time, WCA has contractually committed to apply formal methods to the safety-critical parts of the BATS system, WCA may optionally – that is, not contractually – apply formal methods to other parts of the system. Such an investment may reduce project risks for the entire system.

In the BATS system, the customers have requested that object-oriented techniques be used. However, if they had not, the proposed solution would still apply. At this moment, there is no consensus on the best specification technique that lends itself to object-oriented design and implementation. While the threads-based technique may not be the ideal technique conducive to object-oriented design, they are a good match. Using object-oriented techniques would imply that software engineers with no prior experience in these techniques would have to be trained in object-oriented practices.

If WCA had no interest in staying in the same line of product, the proposed approach is still the most appropriate for the BATS system. While the modifiability of the Requirements Specification may not be as critical then, it is nevertheless important that the document be useful to all parties involved in the product development. The readability of natural language, the possibility of systematic derivation of the specification, the ease of automatic analysis are equally important in specification of a product even though its Requirements Specification may not be reused for other similar products.

However, if the defined system was much smaller and less complex, then Structured Analysis may be used to specify a full model of the system. With a smaller system, the requirements are simpler to understand and the tasks of design and test engineers are not as overwhelming. Of course, the threads-based technique is as suitable as Structured Analysis for such a system. The customers will have to be trained to understand the graphical notations of Structured Analysis so that they are able to assist in requirements reviews. Even if an organisation wishes to stay in the same line of products, it should be possible to redo the models using Structured Analysis for each application provided the system is not too complex. Knowledge of the domain can be used to fully understand and address individual customers' needs. Therefore, for small systems, using only one technique is enough. In any case, the budget for small systems is such that tools and training for all three techniques used for the BATS system cannot be afforded. For small systems, the requirements can be specified directly using any of the four techniques because of the relative simplicity of the systems.

If systems are as complex as the BATS system, then it is imperative that the specification technique inherently has guidelines to facilitate the specification process and to impose some structure on the document. As it is, the complexity of the system may be such that it can be quite overwhelming. In such cases, precautions should be taken so that the technique increases the chances of deriving an adequate specification.

If the system is dominated by safety-critical requirements, then formal methods is the most appropriate specification approach since they may reduce project risks by ensuring that all the requirements are captured and properly understood. Customers will have to weigh the tradeoff between increased confidence in the system and expenses incurred to train engineers in the techniques.

Overall, for any system size or complexity, it is important to choose the specification approach that not only has an adequate notation but that also helps in the production of a truly useful Requirements Specification. It is equally important for the specification approach to help properly analyse and understand customer requirements.

Bibliography

[1] B. Boehm. Verifying and Validating Software Requirements and Design Specifications. *Computer*, 1, January 1984.

[2] Cadre Technologies Inc. *teamwork/SA- teamwork/RT User's Guide.* Cadre Technologies Inc., Providence, Rhode Island, 1991.

[3] Don Chandler. Requirements Analysis and Specification. Software Productivity Centre, Vancouver, B.C., 1993.

[4] Don Chandler, June 1994. Interview – thesis related.

[5] Dave Clark, June 1994. Interview with Dave Clark of MacDonald Dettwiler Associates – thesis related.

[6] comp.specification.z. An Internet newsgroup dedicated to discussions on Z and Z variants.

[7] Dan Craigen, Susan Gerhart, and Ted Ralston. Volume 1: Purpose, Approach, Analysis, and Conclusions. In *An International Survey of Industrial Applications of Formal Methods*, March 1993.

[8] Dan Craigen, Susan Gerhart, and Ted Ralston. Volume 2: Case Studies. In *An International Survey of Industrial Applications of Formal Methods*, March 1993.

[9] Nancy Day. A Model Checker For Statecharts (Linking CASE Tools With Formal Methods). Master's thesis, University of British Columbia, Vancouver, 1992.

[10] Tom DeMarco. *Structured Analysis and System Specification.* Yourdon Press, Prentice-Hall Inc., New Jersey, 1979.

[11] Department of Defense. DOD-STD-2167A – Military Standard, Defense System Software Development. Military Standard, June 1988.

[12] Michael S. Deutsch and Ronald R. Willis. *Software Quality Engineering – A Total Technical and Management Approach.* Prentice Hall Series in Software Engineering, Englewood Cliffs, New Jersey, 1988.

[13] Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z Specification Language: Version 1. Technical report, The Department of Computer Science, University of Queensland, 1991. No. 91-1.

[14] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions On Software Engineering*, 16(4):403–414, April 1990.

[15] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House Publishing, New York, New York, 1988.

[16] Hughes Aircraft of Canada Ltd. SRS Document Construction and Writing Style Standards. In *Software Requirement Specification Policies, Plans, and Procedures (SP3)*, pages 2–1–2–69, August 1993.

[17] D. C. Ince. *An Introduction to Discrete Mathematics, Formal System Specification, and Z.* Oxford University Press, Inc., New York, 1992.

[18] R.B. Jones. ICL ProofPower. In *BCS FACS FACTS*, volume 1, 1992.

[19] Jeffrey Joyce. Formal Specifications Techniques and CAATS: The Results of a Preliminary Investigation. Report written for HCSD as result of preliminary investigation. With contributions from M. Donat, S. Kahan, H. Wong and Z. Zhu., December 1993.

[20] Jeffrey Joyce, May 1994. Personal Communication.

[21] Jonathan Masters, June 1994. Interview with Jonathan Masters of Hughes Aircraft of Canada Ltd – thesis related.

[22] Trevor Paine, Philippe Kruchten, and Kalman Toth. Modernizing ATC Through Modern Software Methods. In *Proceedings of the 38th Annual Air Traffic Control Association*, October 1993. Joint paper of authors from Hughes Aircraft Canada, Transport Canada, and Rational.

[23] David Parnas and Paul Clements. A Rational Design Process: How and Why to Fake it. *IEEE Transactions On Software Engineering*, 12(2):251–257, February 1986.

[24] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z.* Prentice Hall International Series in Computer Science, University Press, Cambridge, UK, 1991.

[25] Wayne Powell, June 1994. Interview with Wayne Powell of Software Productivity Centre – thesis related.

[26] Roger Pressman. *Software Engineering – A Practitioner's Approach.* McGraw-Hill, Inc., 1992.

[27] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[28] Graeme Paul Smith. *An Object-Oriented Approach to Formal Specification.* PhD thesis, The Department of Computer Science, University of Queensland, 1992.

[29] Ian Sommerville. *Software Engineering.* Addison-Wesley Publishing Company, 1992.

[30] J. M. Spivey. *The fuzz manual.* Computing Science Consultancy, York, U.K., 1993.

[31] Mike Spivey. A Guide to the zed Style Option, December 1990.

[32] The Institute of Electrical and Electronic Engineers, Inc. ANSI/IEEE Std 830-1984 – IEEE Guide to Software Requirements Specifications. American National Standard, February 1984.

[33] P. Ward and S. Mellor. *Structured Development for Real-time Systems.* Prentice–Hall, Englewood Cliffs, New Jersey, 1985.

Appendix A

Ad hoc Natural Language Specification

An *ad hoc* natural language specification
of the SIS-TELEREG system
(Version 1.6)
Marie Hélène Wong Cheng In
February 17, 1994

A.1 Introduction

A.1.1 Purpose

This document is written to specify the requirements of the **Telereg**[1] system which has been in existence since 1988. It is expected that the use of this document will be valuable in the maintenance of the system. This document will only be concerned with the functionality of a subset of the **Student Information System** that is related to the **Telereg** system.

A.1.2 Background

Before the **Student Information System (SIS)**, front-loaded with the **Telereg** system was installed, registration at the University of British Columbia, UBC, was done manually. In the pre-1988 days, registration took place during the week after Labor Day at the War Memorial Gymnasium, UBC. Each student would pick up an "*authorization to register*" form and collect course cards or signatures from department representatives for each course she wished to enrol in. The student would then have her schedule approved by her school or major department. The "*authorization to register*" forms would then be returned to the Registrar at the gymnasium. Each night of registration week, the course cards would then be fed into a card reader to be loaded into the **Student Record System (SRS)**. The **SRS** was an archaic system that was unable to meet the growing needs of the Registrar's office and the University.

The **Student Information System** is the new system at UBC that is designed to establish a single database for student information. It allows decentralised update and inquiry access with appropriate access controls, and provides convenient interfaces to other centralised administrative systems. The **Student Information System** also provides data to support comprehensive reporting and analysis as well as the flexibility to implement new policies and support changing needs of the University, faculties and departments. **Telereg** is UBC's telephone registration system which was introduced in

[1] *TELEREG$_{TM}$* is a registered trademark of the University of British Columbia (UBC).

1988. Telereg is the student's interface to the **SIS**. Telereg allows students to register for their courses from any touch-tone telephone. The students use the buttons on the telephone keypad to enter requests and the computer's voice guides them through their registration.

A.1.3 Sources of Information and Acknowledgements

Sources of Information

- Doug Loewen, Senior Technical Analyst, Administrative Technical Support, University Computing Services, UBC.
- Maureen Elliott, Administrative Supervisor, Records and Registration, Office of the Registrar, UBC.
- Beth Smith, Programmer Analyst, Production Systems, University Computing Services, UBC.
- IBM 3270 Information Display System Component Description GA27-2749-10.
- IBM Dialog manual.
- The University of British Columbia 93/94 Calendar
- The University of British Columbia 93/94 Registration Guide
- The VOCOM-40 brochure from Perception Technology, USA.
- Elaine Wright, B.C.Tel Representative for UBC.

Acknowledgements

I wish to thank Maureen Elliott, Beth Smith, and Doug Loewen for their invaluable time and their willingness to meet with me on numerous occasions to explain the intricacies of the SIS-Telereg system. They have been unselfish with their time and have provided me with useful information. Elaine Wright is also to be thanked for her help in the technical aspects of telephone communications. My gratitude goes to Carol Whitehead and Jean Forsythe who have helped me in their own ways to understand the SIS-Telereg system better.

A.1.4 Organization

The established context for this project consists of physical, peripheral, hardware, and software components described in Section A.2. Section A.3 specifies the functional requirements of the system while Section A.4 presents the non-functional requirements of the system. Section A.5 specifies the kinds of exceptional events that may occur and how the system responds to them. Section A.6 describes how the system is expected to evolve in the future. Section A.7 specifies the method used to validate the system. A glossary of technical terms used in this document is found in the last section.

A.2 Context

The established context for the project includes all of the physical, peripheral, hardware, and software components of the Telereg system. Section A.2.1 of this document describes the physical aspects of the established context. The peripheral and hardware contexts are specified in Section A.2.2, and Section A.2.3 provides a detailed description of the immediate context of the project, that is the software interface.

A.2.1 Physical Context

The physical components of the established context consists of a touch-tone telephone and a business line with an overline service in which calls are automatically completed to the next available line of a group of 48 lines when the dialed number is busy.

Touch-tone Telephone

The touch-tone telephone keypad and receiver provide the input and output devices respectively to the student. Each button on touch-tone phones makes a different tone (as specified by some telecommunications standard) when pressed. This allows combinations of sequences of touch tones. Verbal messages are used to request for information or to provide feedback on execution of request.

Business Line

The business line has a maximum of 48 overlines. Telereg offers one business telephone number to call and the telephone company distributes that one phone line to the first free overline. When there are too many calls from an area, they are blocked by the telephone company at its discretion even though not all the lines may be used.

A.2.2 Peripheral and Hardware Contexts

The peripheral components of the established context consists of a VOCOM-40 voice processor and a protocol converter while the hardware component consists solely of an IBM 390 **Multiple Virtual System** (MVS) mainframe.

Figure A.59 shows how the various components of the peripheral and hardware contexts are linked together. The voice processor interfaces directly with the IBM MVS machine through the protocol converter. The functions of each of these components will be explained in the following three sections.

VOCOM-40 Voice Processor

The main two components of the VOCOM-40 system are the BT-III Voice Processor and the Digital Speech Recorder (DSR). An additional component, the Application Processor (A/P) will be installed and Section A.6.2 on page 212 describes how its addition to the existing configuration can enhance the system.

The BT-III Voice Processor is a DEC PDP-11, model 73 machine. The BT-III Voice Processor is responsible for phone communications, touch-tone recognition, and speech. It operates 48 phone lines. The model has a capacity of 48 lines within one box and can be linked with another BT-III box for a total expansion to 96 lines. The BT-III

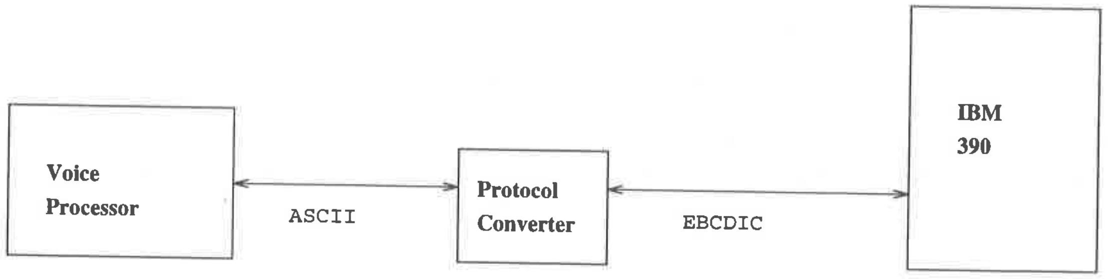


Figure A.59: Linking the peripheral components to the hardware component

is currently configured with a capacity of 30 hours of speech and is expandable to 390 hours. It is capable of sending or receiving voice messages. Incoming messages can be recorded like an answering machine or like voice mail. Each outgoing message is digitised and stored on disk, and is referenced by a unique "message number". Individualised messages that are transmitted to the caller are composed at run time by assembling together shorter messages that are referenced by their unique "message number". The software on the IBM specifies the message numbers of the shorter messages that will compose the complete message and the order in which they will appear. The Voice Processor composes the message and outputs it verbally.

The BT-III supervises calls and is capable of transferring or initiating calls. The Voice Processor has some built-in commands such that it can pick up the phone, hang up the line or interrupt its speech if touch tones are received. However, it will execute such commands only when and as directed by the software running on the IBM machine. The IBM indicates that a command is to be executed through a character and possibly a line number. The BT-III has built-in logic that associates the character with an action. When a user calls the business line, the BT-III detects the ring and informs the IBM. The software on the IBM 390 machine will determine whether to answer the phone and will pass on the letter command to execute to the BT-III and the line number. As such, the Voice Processor knows nothing of the Telereg application. It can give out messages and accept input from the telephone. It also converts the touch tones to ASCII data stream.

The Digital Speech Recorder is used to record and edit speech, and download speech to the BT-III.

Details of the protocol between the Voice Processor and the protocol converter are given in the brochure distributed by the manufacturer of the Voice Processor on the VOCOM-40. The connection between the Voice Processor to the protocol converter is a Null Serial Cable.

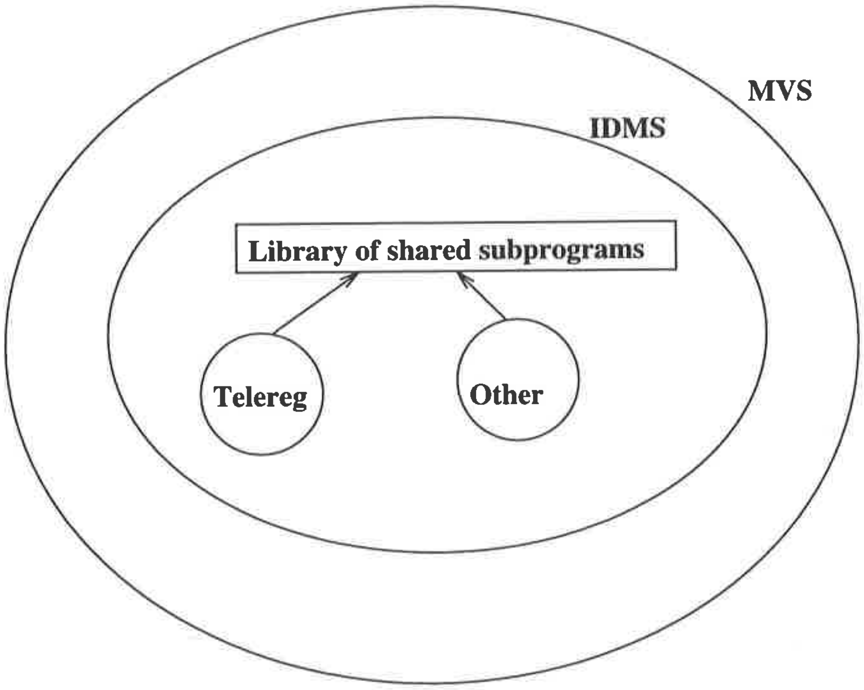


Figure A.60: Software context

IBM 390 Mainframe

Both the SIS and Telereg system software reside on an IBM 390 MVS mainframe. There are 48 communication channels on the IBM reserved for the Telereg application. Multiplexing between the Voice Processor and the IBM is based on the line number that is generated by the Voice Processor. Data from one channel on the IBM is made into a packet which includes the line number provided by the channel. The connection between the IBM to the protocol converter is Ethernet.

Protocol Converter

The protocol converter is actually a program that runs on a PC. It converts ASCII data stream from the Voice Processor to an EBCDIC datastream in 3270 format, as specified in document GA27-2749-10, IBM 3270 Information Display System Component Description, to the IBM machine, and vice versa. The protocol converter, in essence, allows the Voice Processor to communicate with the hardware component. Since there are 48 possible lines in the Voice Processor, the protocol converter checks the line number associated with the data in order to send the converted data to the appropriate channel at the IBM or telephone connection depending on the direction in which the data came from. Communication over the protocol converter is full duplex and the rate of conversion is limited by the speed of the line. It converts data either way and does one conversion at a time. Data from the Voice Processor is queued at the Voice Processor and data from the IBM is queued at the IBM if there is contention for data conversion from either side. Data from the protocol converter is meant to look like terminal I/O to the IBM channel.

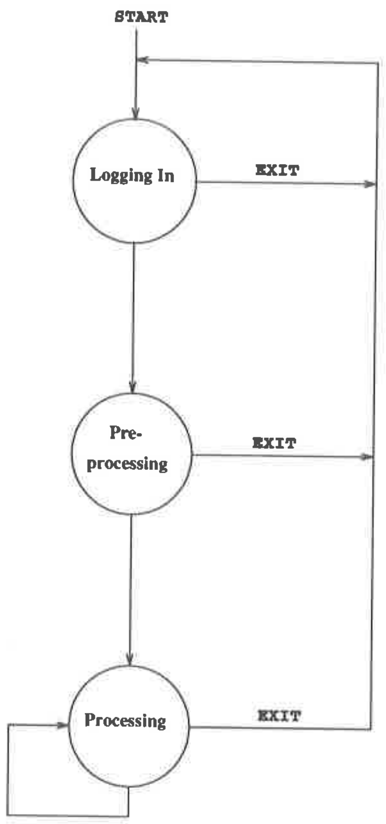


Figure A.61: Modes of operation

A.2.3 Software Context

The IBM 390 mainframe runs the **Multiple Virtual System** (MVS) operating system which runs the product **Integrated Database Management System** (IDMS), an operating system for database management. Within IDMS, various SIS applications including Telereg are run (see figure A.60). These applications share the same database and some of the same subprograms. The only distinguishing feature between these applications is their front-end programs. The front-end programs handle the input and output data and call the appropriate subroutines to execute the request. The front-ends programs do not interact with each other. Telereg gives students the means to access a limited set of functions of the SIS system. When IDMS is started, all applications are started as well.

A.3 Functional Requirements

The SIS-Telereg telephone registration system has three main modes of operation: "Logging In", "Preprocessing", and "Processing".

Figure A.61 shows the transitions from one mode of operation to the next. The "Logging In" mode is always the starting mode of operation. From the "Logging In" mode, the telephone registration system can either proceed to the "Preprocessing" mode or exit. From the "Preprocessing" mode, the system can move on to the "Processing" mode or exit. Finally, from the "Processing" mode of operation, the system can either exit to the starting mode of operation or go back to the same mode of operation.

In each of these modes, the user may request for the previous message to be repeated.

A.3.1 Logging In

In the "Logging In" mode of operation, the student identification is validated and it is checked whether the student has financial holds or disciplinary blocks on her record, as well as whether the student has an outstanding fee payment.

Validating Identification

At the start of each registration session, the student is asked for her student identification. If the student identification is valid, then the first four letters of the student's last name are spelled out verbally. The student identification has been validated. If either the student identification format or the student number or the birth date is invalid, the student is informed that there is some kind of error in the data provided and she is requested to provide her student identification again. On her third unsuccessful attempt at entering the registration process, she is advised verbally by the system to seek outside assistance and the system terminates the telephone connection.

Withholding Student Registration

If the student's record shows any disciplinary blocks or financial holds placed on it, then the student is not allowed to register. She is advised to contact the appropriate department or administrative office for more information. The student will be withheld from registering. This check is done when the student gives her student identification.

Checking for Outstanding Fee Payment

The third check that is done when a student provides her student identification is on her fee balance. If the student has an outstanding fee balance, she is reminded of it and the system will only allow the student to ask for transactions such as fee balance or section openings.

Exiting the "Logging In" Mode of Operation

The student will have to successfully pass the above three checks when she provides her student identification before she is allowed to proceed further in the registration process to the next mode of operation, "Preprocessing". If the student identification is invalid or the student's record has disciplinary blocks or financial holds placed on it, then the system will exit the current mode of operation.

A.3.2 Preprocessing

In the "Preprocessing" mode of operation, the software checks whether the student is qualified to access Telereg and whether she is eligible to register for courses. In this mode of operation, the student is also allowed to enquire about her current fee assessment.

Computing the Current Fee Assessment

Even though a student might not be allowed to register for a session, she is allowed to query her current fee balance. If the student does not provide a session, the default session is the previous session. The system computes the current tuition and fees balance owed by the student based on courses she has enrolled in for the session and the payments already made to the university. The dates at which payments are due are also given.

Checking the Access Dates

Each student is assigned an access date based on her year, degree program, category, previous year session average or admission average. That access date is the first day that the student can access a particular session via Telereg. When the student provides the session for which she wishes to register, the system checks whether the student should be allowed access to the registration process yet. If that is not the case, the system informs the student that she cannot access Telereg yet and terminates the telephone connection.

Checking for Eligibility to Register

A student has an eligibility to register if she is a student newly admitted to the university or she is a continuing student whose academic performance in the last session meets university requirements for continuing an academic career at the university or she is a continuing student who has been approved for a program by a department. When a student provides the session for which she wishes to register, her eligibility to register is verified. If the verification fails, then the system informs the student of her ineligibility to register and terminates the telephone connection. If the student has not specified a specialisation or a program and one is needed, the system requests for the specification or the program. The system verbally gives the specialisation or program that is on file for the student.

Exiting the "Preprocessing" Mode of Operation

A student may proceed to the "Processing" mode of operation if she is accessing Telereg on or after the access date assigned to her and she is eligible to register for the session. Otherwise, the system exits the current mode of operation.

A.3.3 Processing

In the "Processing" mode of operation, the student is able to actually register for courses provided that the requests are in the proper format and are valid. This validity check is done before the registration request is executed. The system does not allow concurrent updates of a student's record.

Checking the Validity of Requests

When the student enters a registration request, the format of the request is checked for validity. If it is not valid, the student is informed of the error in the data and is asked to provide the command again. If a timeout occurs before a complete request is made, the student is invited to enter her request again unless the maximum number of timeouts has been exceeded. If the student has more than zero but fewer than ten free transactions left, she is informed of the exact number left.

Except for the repeat message request and the exit request, before any request is executed, it is checked whether the student's record is being updated by another user. If the check is positive, then the system advises the student to try at a later time and terminates the telephone connection.

Executing the Exit Request

If the request is a command to exit the registration session, the system calculates the student's current fee assessment based on previous and current transactions made to the student's schedule, gives a message about the amounts and dates by which payments are due, and then terminates the telephone connection.

Charging a Fee for the Transaction if Necessary

Upon receiving a request that has been validated, if the request is any one of adding a section, dropping a section, changing a section, or looking up an inquiry on a section and the student has used up all her free transactions, then she is charged a transaction fee that is set by the Registrar's Office. That fee is added to the student's financial account. If the maximum charge fee is exceeded, the student is restricted from registering and the system terminates the telephone connection.

Adding a Section

If the registration request is a command to add a section and the section to be added and the student has already been registered in a Standard TimeTable, she will not be able to make any changes to her schedule. If there are openings in the section and the student will not have exceeded the maximum number of credits that she is allowed to register in with the addition of the section and the student is not already registered in that section, the student is added to the classlist for that section and the number of openings in the section is decremented by one. The section is also added to the student's registration schedule for the session. The appropriate feedback based on the success or failure in adding the section is given verbally by the system. The student's number of free transactions is decremented by one.

Confirming a Section Switch

If the registration request is a command to confirm a section switch and the section to confirm, it cannot be executed if the student has already been registered in a Standard TimeTable. Also, if the student has an outstanding fee balance, the request cannot be executed. Otherwise, the system searches for the specified section in the student's schedule. Based on the results of this search, the system gives the appropriate verbal feedback. The student's number of free transactions is decremented by one.

Changing Section

If the registration request is a command to change a section and the new section to change to, the system gives a negative feedback if the student has already been registered in a

Standard TimeTable. Otherwise, if there are openings in the new section and the old section can be found in the student's schedule, then the old section is changed to the new section in the student's schedule. Besides, the number of openings in the old section is incremented by one while that of the new section is decremented by one. The student is added to the classlist of the new section and removed from that of the old section. The student is informed of the results of the execution of this request. The student's number of free transactions is decremented by one.

Dropping a Section

If the registration request is a command to drop a section and the section to be dropped, the student is not allowed to do so if she is already registered in a Standard TimeTable. Otherwise, if the section can be found in the student's schedule, it is deleted and the student is removed from the classlist associated with the section. The number of openings in that section is incremented by one. A verbal message reporting the results of the request is given to the student. The student's number of free transactions is decremented by one.

Looking up Inquiry on Sections

If the registration request is a command to look up an inquiry about a section and the section to look up, the number of openings in that section is reported. The student's number of free transactions is decremented by one.

Listing All Courses

If the registration request is a command to list all the courses in a student's schedule, the student's schedule for the session is given verbally.

Cancelling Entire Registration

If the registration request is a command to cancel the entire registration and the student has not been registered under the Standard TimeTable, then for each of the sections that the student has registered for, it is deleted from the student's schedule. The student is removed from the classlist associated with each section and the number of openings in each section is incremented by one. Feedback is given to the student.

Computing Current Fee Assessment

If the registration request is a command to compute the current fee assessment, the current tuition and fees balance owed by the student based on courses he has enrolled in for a session and the payments already made to the university is calculated . The dates at which payments are due are also given.

Adding Specialisation

If the request is a command to add a specialisation or a program, the latter is added to the student's record and feedback as to what specialization or program the system has on the student is given.

Exiting the "Processing" Mode of Operation

A student may return to the current mode of operation if the registration request has been successfully executed and the request was not an exit request. The system exits from the current mode of operation if the registration request was an exit request or the maximum number of timeouts has been exceeded.

A.4 Non-Functional Requirements

A.4.1 Product Requirements

Compiled code for the program must fit within the memory size of the program store. The required software is implemented in the ADS/O COBOL language. In addition, the software conforms to the coding standard specified by the customer.

A.4.2 Response Time

Since response time is a continuing concern, it has been monitored since Telereg was first introduced and various performance tuning projects over the years have been initiated to improve program efficiency. Response time is directly related to the amount of processing done by each command and the load on the system. To improve the response time, every program in the Telereg system was tuned and a better mainframe was installed. The system was designed such that the processing load is minimized and no one command does too much processing. For instance, data is not validated more than once and validation occurs only when the data is needed.

On average, a call usually lasts four minutes. If the response time between the student completing a command entry and the response to that command is greater than 60 seconds, then steps are taken to minimize the load. These remedial steps include any of the following: shutting down some of the 48 phone lines, minimizing other activities on the mainframe such as giving development programmers a lower priority for CPU cycles, and restricting the use of certain heavy user commands such as the "Compute Current Fee Assessment" command during the peak hours of the day. Part of the data for the whole **Student Information System** is kept in tables known as **"YES/NO tables"**. By looking up the "YES/NO" value of an entry, a subroutine will either enable or disable the completion of its execution. The "Compute Current Fee Assessment" function checks the table entry associated with it before performing the command. By simply setting the table entry associated with the "Compute Current Fee Assessment" function to "no", its use will be restricted.

A.4.3 Reliability

The software needs to recover gracefully if exceptional events occur. To decrease the probability of failure, three programming environments are used. They are each dedicated for a specific purpose: development, testing, and production. All new software is tested in the testing environment which includes a full-sized database and two Telereg phone lines. The new software needs to be approved by the Registrar's Office before it goes into the production environment.

A.4.4 Security

All administrative systems, including the SIS and Telereg, are situated on a separate mainframe which is physically isolated from all other UBC computer systems. All connections to this mainframe are in physically secure locations. Identifications and passwords are required to connect to the system and there are several levels of security to control which screens are available to each user. If a screen is not accessible by a user, it will not appear on that user's menu. A further level of security controls which actions are available to a user on a given screen.

The primary security check that the system performs for Telereg callers is on birth dates. In essence, the birth date is a pin number. The student may request that the Registrar's office change her pin number. The software will check that the given pin number matches the pin number on file for that student.

All changes to data are logged to both SIS specific database save areas and IDMS specific journal files which are later copied to tapes.

A.4.5 Concurrency

Concurrency is achieved by running 48 instantiations of the Telereg front-end programs for each channel. This implies that, at any one time, the Telereg system can support up to 48 simultaneous users which include both students and Registrar personnel. This maximum number of users can be increased by getting a greater number of overlines from the local telephone company and running more instantiations of the Telereg front-end program.

A.5 Exceptional Events

Hardware overheating, hardware component failure, application software bugs, and telecommuncation failures have been some of the reasons causing the Telereg software to crash over the years. Since crashing software is inevitable, the Telereg software has been designed such that, in the best case, when the software detects a problem it tells the student "There has been a Telereg system error. Please call again later. Goodbye." In the worst case, the software crashes and the telephone connection is terminated.

To ensure that the software recovers gracefully, if the Voice Processor is still running, it enables an "application down" routine and answers the lines with an advisory message and then terminates the telephone connection. At the time of a crash, all completed commands from a given phone session will have been processed and saved, but commands in progress may not have taken effect. It is up to students to check their last command when they phone back.

To minimize downtime in the event of problems, various spare hardware components are on site. Operators who are available at all times to monitor the Telereg system have a list of people to contact in case problems occur.

Two common exceptional events occur when a registration session is not terminated by the exit sequence or the touch tones are fuzzy. In the former case, the voice response unit detects such events as hangups and inactivity timeouts and notifies the mainframe of these events. The application software is coded to handle a hangup. In the latter case, anything other than proper touch tones would not be understood and would be equivalent to inactivity and result in a timeout condition.

A.6 System Evolution

A.6.1 Software Evolution

It can be expected that the software can be further modified to allow verification of time conflicts in a student's registration schedule. Also, at this time, the software does not check whether pre-requisites and co-requisites are met before allowing a student to register in a course. It is deemed that computer time to check the pre-requisites and co-requisites of each and every course would greatly slow down a telephone registration session by increasing the response time.

A.6.2 Peripheral Evolution

The Application Processor is one of the three components of the VOCOM-40 voice unit that has not been installed yet. The Application Processor is a 486 PC that runs interactive UNIX. Interactive UNIX allows multiple users, multi-tasking, multiple host connections, and multiple applications. The Application Processor is responsible for application control and host communications (if connected to a host). An application generator, ScriptPower, can be installed on the Application Processor and be used to generate voice processing applications via easy-to-use pull-down menus.

After installation, it will be connected between the Voice Processor and the protocol converter. It is expected that, in the long run, applications with telephone access other than Telereg will be preprocessed on the Application Processor before they are properly dispatched to the IBM mainframe to the appropriate application program. In the case of the Telereg application, the addition of the Application Processor will alleviate time resource requirements on the mainframe because most validations on data content and format can be performed on it.

A.7 Validation Criteria

The customer will provide an acceptance test in the form of a series of executions of the program under varying conditions and test cases to test its functionality and its limits.

A.8 Glossary

Action command - A capital letter that indicates the type of transaction to execute. The set of allowable action commands is R, T, A, C, D, I, L, M, X.

Disciplinary Blocks - These are restrictions that are placed on a student's record to prevent her from registering. These restrictions arise because of academic reasons such as failure to keep in good academic standing and so on.

Financial Holds - These are restrictions that are placed on a student's record to prevent her from registering. These restrictions arise because of financial reasons such as an outstanding financial balance or any other reasons that would require for the student to clear with the financial departments first before she is allowed to register.

Free Transactions - Use of Telereg is free for the first 60 transactions. A student should be able to complete her registration within this arbitrary number of transactions that is determined by the Registrar's Office. Once that number is exceeded, the student is charged a fee for each of her transactions. Beyond a maximum charge fee, the student is restricted from registering.

Section number - A unique 5 digit number that corresponds to a course and its section. The valid section numbers can be found in the catalog.

Session code - Used to indicate the session for which the student wants to register. It consists of a 2-digits number to indicate the academic year and a capital letter to indicate the semester. There are two allowable semesters: summer (S) and winter (W). The summer session spans over the months of May to August of the same year whereas the winter session spans over the months of September of one year to April of the next year.

Specialisation - A 4-digits number to indicate the program or specialisation of the student. The valid values can be found in the university catalog.

Standard TimeTable - A number of course sections that have been grouped together within a section number to form a conflict free registration package required for a specific program. Students are automatically registered in each of the courses in a Standard TimeTable if they belong to a specific program.

Student identification - Made up of a unique 8-digits student number and a 6-digits birth date in the form YYMMDD where YY represents the year, MM the month and DD the day.

Appendix B

Structured Analysis Specification

A Teamwork Structured Analysis
Specification Model of the SIS-Telereg System
Marie Hélène Wong Cheng In
September 6, 1993

B.1 History

Created September 6, 1993.
Updated February 21, 1994 after review by Jeff Joyce.
Updated August 23, 1994.

B.2 Introduction

This document is a proposal for a model of the **Student Information System-Telereg System** at the University of British Columbia, UBC. Telereg is the University's telephone based registration system. The Student Information System (SIS) is the new system at UBC that was designed to establish a single central database for student information. This model of an already existing system is primarily concerned with a subset of the Student Information System that is related to the registration process. The model depicts the registration process be it through the telephone or through a terminal. The model ignores the form of data for registration but rather focuses on its contents to show how it is transformed from input to output. The model was developed with the use of **Teamwork SA/RT CASE** tool (Version 4.1) in a top-down fashion starting with a context-diagram to several levels of data flow diagrams. It is assumed that the reader of this document is familiar with the features and notation used in Structured Analysis and, in particular, Teamwork SA/RT (version 4.1).

B.3 Model Checking

The model specified in Teamwork SA/RT CASE tool has been checked using the facilities offered by the tool. The properties that can be checked are: the DFD syntax, balancing of child processes, CSpecs, DDE expansion up to three levels or unlimited levels. A user can either specify to check a diagram only or the diagram and its subtree. The checking tool reports on unmatched CSpec flows, CSpec balancing errors, DDE errors, DFD and PSpec balancing errors, syntax errors in PSpecs, store-to-flow errors, syntax errors in general, errors in states, events and actions expressions, and other syntactic errors. Overall, Teamwork primarily checks the syntax of the model.

To ensure that the specified model actually makes sense, I manually checked the contents and components of the model. I manually checked most of the properties that the CASE tool enables. I have also checked that the dataflows in and out of primitive processes are all actually used in one way or the other in the body part of the process specifications and that they are used in a meaningful way. I have also checked that I have captured all the areas of interest in my model. Since Teamwork does very little checking of control specification objects, I have checked that they are used appropriately and in a meaningful manner. To check that I have a correct picture of the model, I had to understand how the system works and I talked to the customers and system developers of the system (Doug Loewen, Beth Smith, and Maureen Elliot) to reinforce that understanding.

B.4 Overview

Registration at the University of British Columbia is done primarily through the telephone. However, some university administrative personnel from various offices have direct access to the system to register on behalf of students. While the form of data to the system for registration purposes is different depending on who is initiating a registration session with the system, its content is the same. This model depicts the registration system regardless of its user.

The system interacts with a number of data stores. This model shows the kinds of information that are contained in these data stores and how they are updated. The data stores are used for purposes other than registration, e.g. generation of reports. This model shows some data stores being updated but which are not used for any purposes in this model since their other uses are beyond the scope of this model. We have included enough information in the model to show how the registration process fits into the SIS system.

Only some of the users of the SIS system are considered. They are the Registrar's office, cashier office, admissions office, academic departments, schools and faculties. They are of interest because they provide and receive information related to our model of the SIS-Telereg system.

B.5 Maintainance of the Model

If the model of the system is to be updated or changed, the changes are to be made in the model created using Teamwork SA/RT CASE tool. The various objects and elements of the Teamwork model have to be checked for completeness before they are incorporated in this document. Section B.3 describes the kinds of checking that need to be done on the model.

B.6 Sources and Acknowledgements

B.6.1 Sources of Information

- Tom DeMarco. *Structured Analysis And System Specification.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

- Derek J. Hatley, Imitiaz A. Pirbhai. *Strategies For Real-Time System Specification.* Dorset House Publishing Co., Inc., New York, NY, 1988.
- Teamwork user's guides.
- Doug Loewen, Senior Technical Analyst, Administrative Technical Support, University Computing Services, UBC.
- Maureen Elliott, Administrative Supervisor, Records and Registration, Office of the Registrar, UBC.
- Beth Smith, Programmer Analyst, Production Systems, University Computing Services, UBC.
- IBM 3270 Information Display System Component Description GA27-2749-10.
- IBM Dialog manual.
- The University of British Columbia 93/94 Calendar
- The University of British Columbia 93/94 Registration Guide
- The VOCOM-40 brochure from Perception Technology, USA.
- Elaine Wright, B.C.Tel Representative for UBC.

B.6.2 Acknowledgements

I wish to thank Maureen Elliott, Beth Smith, and Doug Loewen for their invaluable time and their willingness to meet with me on numerous occasions to explain the intricacies of the SIS-Telereg system. They have been unselfish with their time and have provided me with useful information. Elaine Wright is also to be thanked for her help in the technical aspects of telephone communications.

Special thanks go to Eric Borm, fellow graduate student, for his useful comments on many versions of the model and for his insight on how to do structured analysis well. The software engineering students that I have taught over the summer of 1993 at UBC also deserve recognition because they have shown me through their various models of the same system how Structured Analysis can best be done. They have given me the platform on which to test my understanding of the methodology. They have shown me some of the common difficulties encountered by novices in the methodology so that I can have a better appreciation of what to be careful about when doing Structured Analysis. Some of them have also taught me better ways of doing Structured Analysis. My gratitude goes to Carol Whitehead and Jean Forsythe who have helped me in their own ways to understand the SIS-Telereg system better. Finally, many thanks to Rob Scharein and Christopher Healey who have been of great help in my struggles with including PostScript files in my LaTeX document so that they can be readable and visible.

B.7 Organisation of Document

This document is organised such that each section is devoted to a level of decomposition in the model. Hence, Section B.8 introduces the system and its environment. The first level of decomposition is described in Section B.9 whereby the system is broken down into four processes, three of which are decomposed further and described in the next three Sections (Sections B.10, B.11, and B.12). The data flow diagram in Section B.12 deals with the registration process and is decomposed to another level which is described in Section B.13. Section B.14 provides further details on one of the processes in the previous section. The appendix contains the complete list of data dictionary entries for the model.

B.8 Context Diagram

The context diagram, shown in Figure B.62, depicts the system and its environment. The external agents that interact with the system are the Registrar's office, cashier office, admissions office, academic departments, schools and faculties, and students. They are the net originators and receivers of data to and from the system respectively. The three primary functions of the system are to update the database on students and faculty information, to allow for student registration of courses either by a student or an administrative employee, and to generate reports at the requests of the various administrative offices. These functions have motivated the organization of the data flows such that they are database information, report requests, report screens or printouts, registration requests, and registration feedback. A user identification is also a necessary data flow because there are levels of security in the system that correspond to varying levels of access to the system.

B.9 Level 0 - Student Information System

The system can be divided into four processes at the first level of decomposition (see Figure B.63) since the SIS system has three main functions and a fourth process is needed to determine the security access level for a given user of the system. This division also satisfies the purpose of the document which is primarily a model of the registration process within the SIS. Thus, processes 1 and 2 will be expanded to at most one more level of decomposition because they do not directly relate to the registration process. PSpec 4 describes how the security access level is determined since process 4 is not further decomposed. Process 3 which deals with the registration process will have three more levels of decomposition.

Process 1 takes database information from various sources and updates some data files. This might raise the question of whether a process is really needed here since updating a data file would hardly qualify as processing and one could easily draw the data flows as going into the data files directly. The question arises because the SIS does not do much processing except for computation of fees or grades for example. However, given that the system is by nature a central database, we find that most of its functions is to update data files and, as such, justifies the use of processes to describe additions, deletions, and updates to the database.

We also show at this level how data stores that have been updated by process 1 are used by other processes (2 and 3) for generation of reports and execution of registration

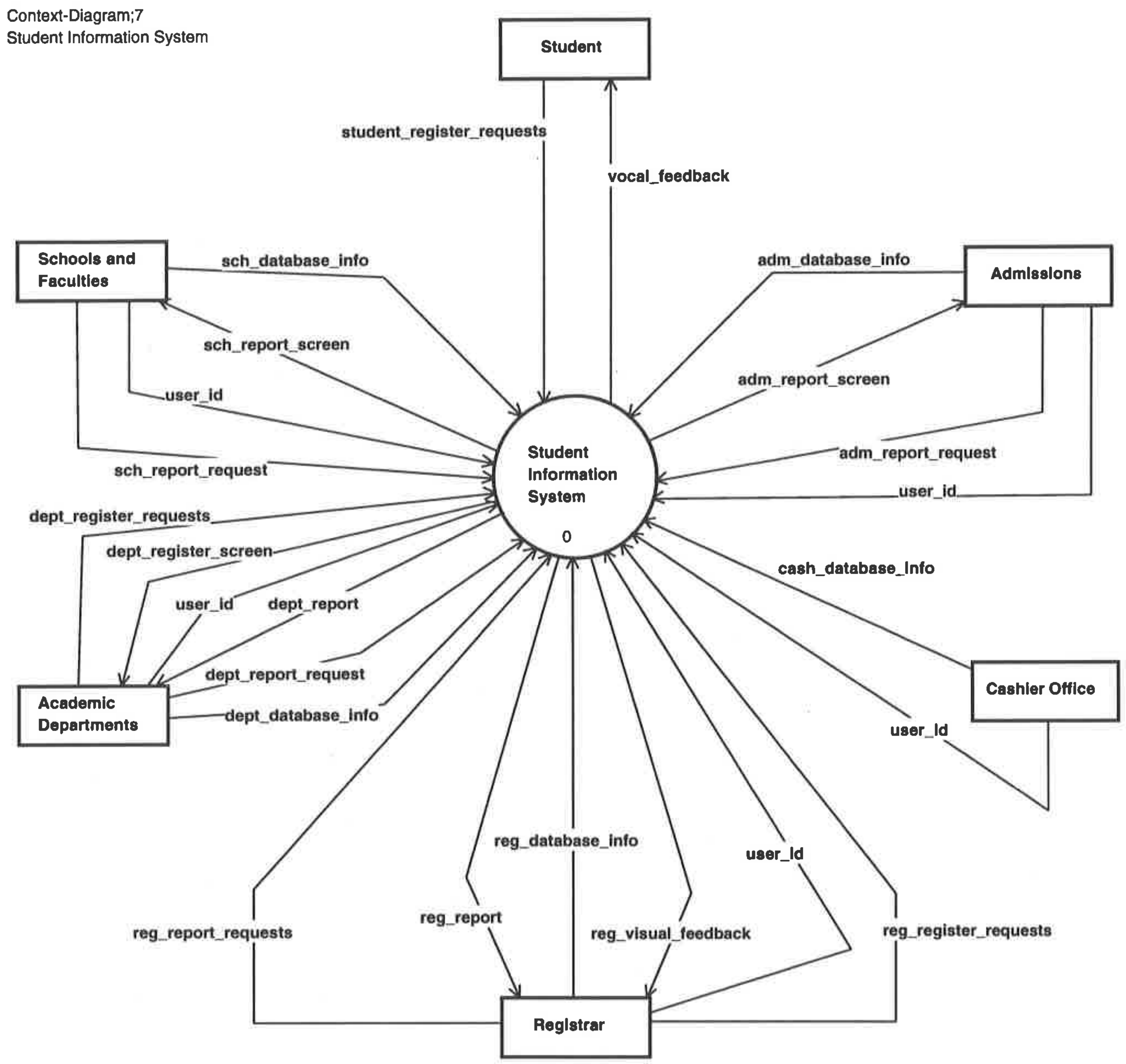


Figure B.62: Context Diagram

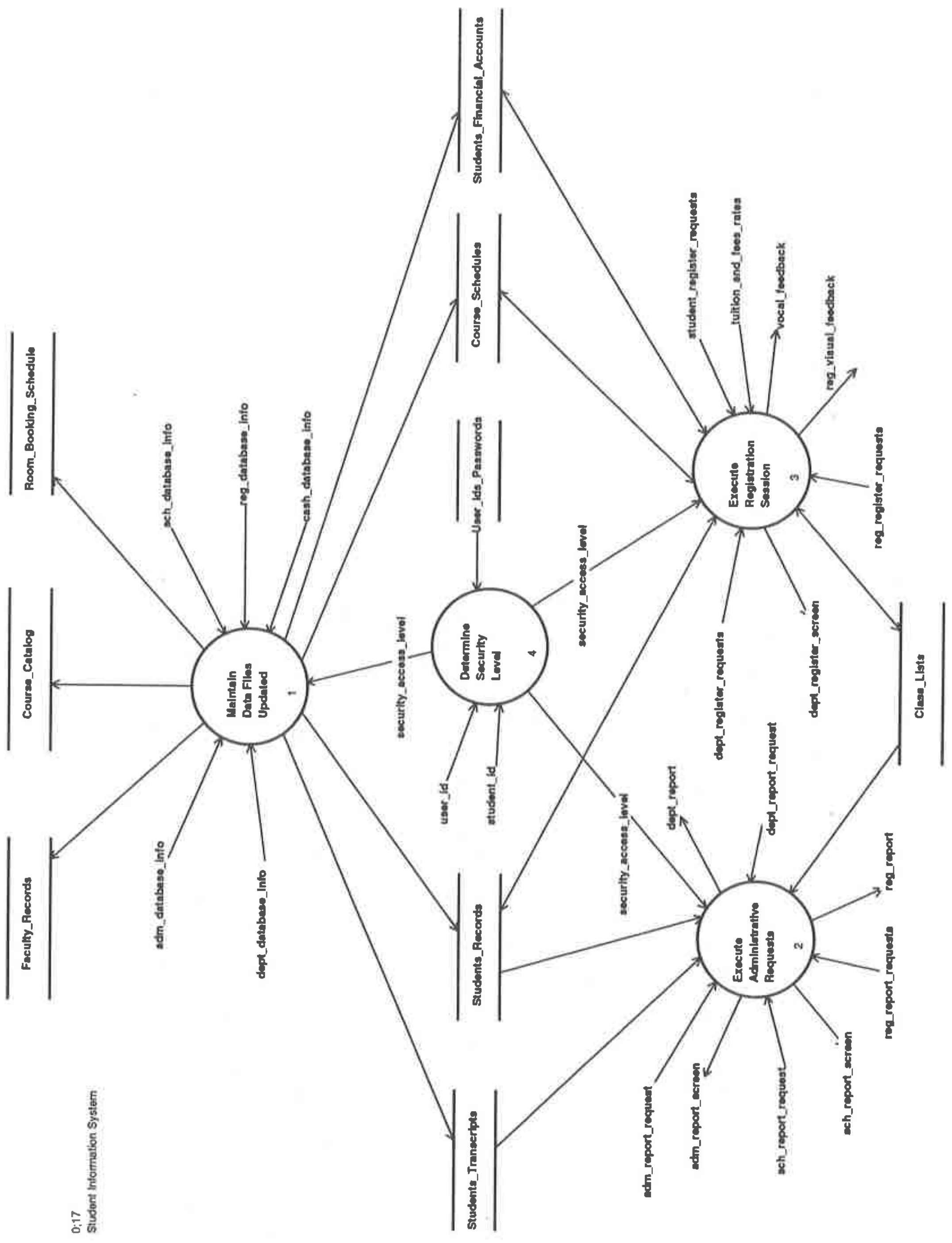


Figure B.63: Level 0 - Student Information System

requests.

```
P-Spec 4;4: Determine Security Level

NAME:
4;4

TITLE:
Determine Security Level

INPUT/OUTPUT:
security_access_level : data_out
user_id : data_in
User_Ids_Passwords : data_in
student_id : data_in

BODY:
IF (<user_id>)
   check <user_id> against <User_Ids_Passwords>
   set <security_access_level> to appropriate value
IF (<student_id>)
   set <security_access_level> to appropriate value for registration
     purposes only
```

B.10 Level 1 - Maintain Data Files Updated

Figure B.64 is the child of process 1 in level 0 data flow diagram (see Figure B.63). Figure B.64 shows the data files which are updated with the various types of information provided by the external sources. The PSpecs of the various bubbles indicate how the processing takes place. Since updating data files is not of particular interest for this document, other than to show that data to be used for registration comes from some data store and to show the kind of data that belongs to the data store, we do not describe in details how the data update takes place.

```
P-Spec 1.1;7: Update Students Records

NAME:
1.1;7

TITLE:
Update Students Records

INPUT/OUTPUT:
Students_Records : data_out
disciplinary_blocks : data_in
eligibility_rejection : data_in
```

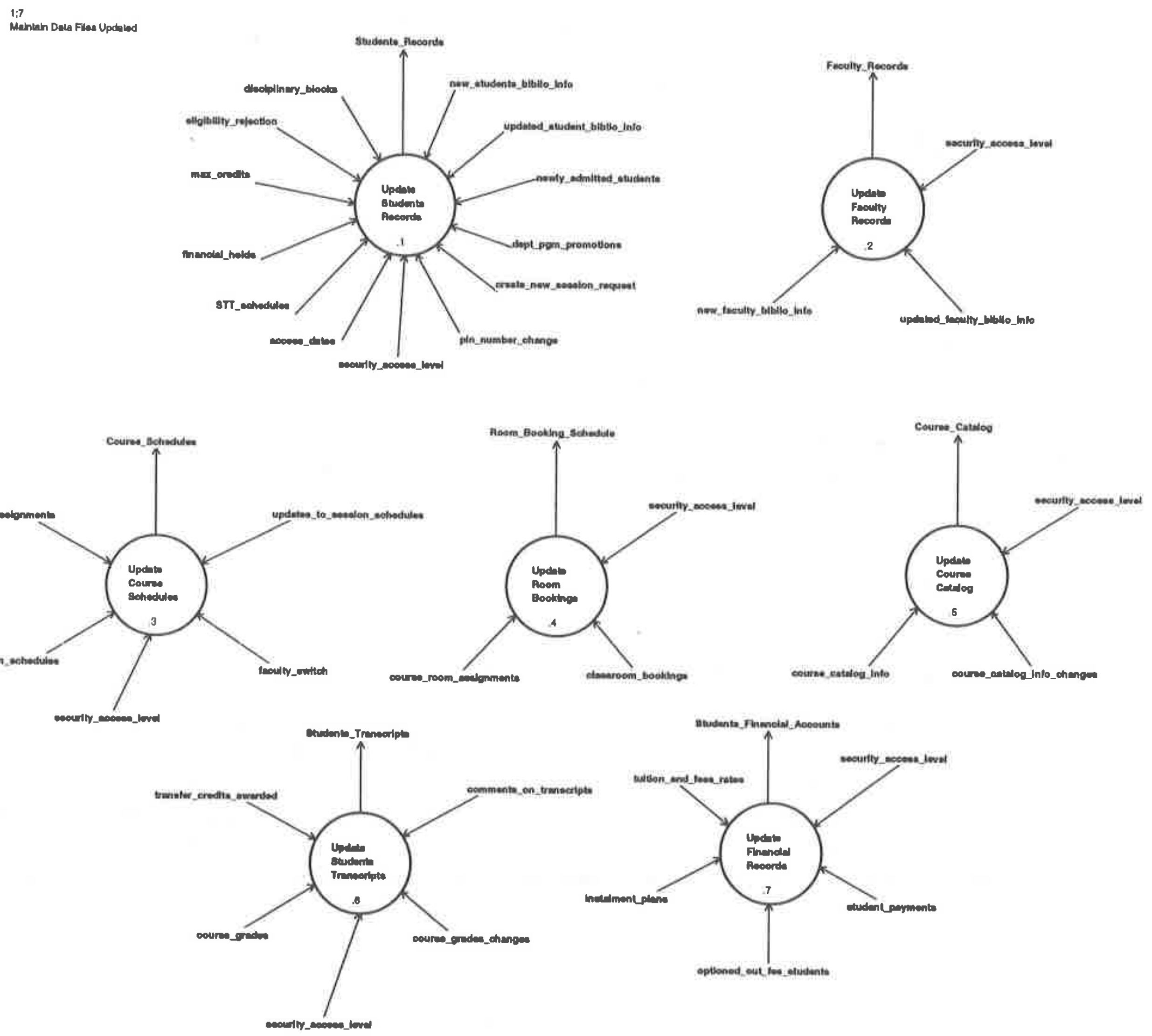


Figure B.64: Level 1 - Maintain Data Files Updated

```
max_credits : data_in
financial_holds : data_in
STT_schedules : data_in
access_dates : data_in
pin_number_change : data_in
create_new_session_request : data_in
dept_pgm_promotions : data_in
newly_admitted_students : data_in
updated_student_biblio_info : data_in
new_students_biblio_info : data_in
security_access_level : data_in

BODY:
IF (<new_students_biblio_info>/<updated_student_biblio_info>
      AND <security_access_level> = OK)
   add to <Students_Records>.biblio
IF (<newly_admitted_students> AND <security_access_level> = OK)
   add to <Students_Records>.biblio
   set <Students_Records>.eligibility = TRUE
IF (<dept_pgm_promotions> OR <create_new_session_request>
      AND <security_access_level> = OK)
   set <Students_Records>.eligibility = TRUE
IF (<pin_number_change> AND <security_access_level> = OK)
   set <Students_Records>.birth_date = <pin_number_change>
IF (<access_dates> AND <security_access_level> = OK)
   <Students_Records>.access_date = <access_dates>
IF (<STT_schedules> AND <security_access_level> = OK)
   <Students_Records>.schedule.STT = OK
IF (<financial_holds> AND <security_access_level> = OK)
   <Students_Records>.hold = TRUE
IF (<max_credits> AND <security_access_level> = OK)
   <Students_Records>.<max_credits> = <max_credits>
IF (<eligibility_rejection> AND <security_access_level> = OK)
   <Students_Records>.eligibility = FALSE
IF (<disciplinary_blocks> AND <security_access_level> = OK)
   <Students_Records>.block = TRUE
```

P-Spec 1.2;7: Update Faculty Records

```
NAME:
1.2;7

TITLE:
Update Faculty Records

INPUT/OUTPUT:
```

```
Faculty_Records : data_out
new_faculty_biblio_info : data_in
updated_faculty_biblio_info : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK)
   add <new_faculty_biblio_info>/<updated_faculty_biblio_info> to
      <Faculty_Records.biblio>
```

P-Spec 1.3;8: Update Course Schedules

```
NAME:
1.3;8

TITLE:
Update Course Schedules

INPUT/OUTPUT:
Course_Schedules : data_out
course_room_assignments : data_in
session_schedules : data_in
updates_to_session_schedules : data_in
faculty_switch : data_in
security_access_level : data_in

BODY:
FOR any of <course_room_assignments> or
              <session_schedules> or
              <updates_to_session_schedules> or
              <faculty_switch>
   IF (<security_access_level> = OK)
       add to <Course_Schedules>.course_room_assignments or
              <Course_Schedules>.session_schedules or
              <Course_Schedules>.faculty_switch
```

P-Spec 1.4;7: Update Room Bookings

```
NAME:
1.4;7

TITLE:
Update Room Bookings
```

```
INPUT/OUTPUT:
Room_Booking_Schedule : data_out
course_room_assignments : data_in
classroom_bookings : data_in
security_access_level : data_in

BODY:
IF  (<security_access_level> = OK)
   add <course_room_assignments>/<classroom_bookings> to
       <Room_Booking_Schedule>
```

```
P-Spec 1.5;8: Update Course Catalog

NAME:
1.5;8

TITLE:
Update Course Catalog

INPUT/OUTPUT:
Course_Catalog : data_out
course_catalog_info : data_in
course_catalog_info_changes : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK)
   add <course_catalog_info>/<course_catalog_info_changes>
       to <Course_Catalog>
```

```
P-Spec 1.6;7: Update Students Transcripts

NAME:
1.6;7

TITLE:
Update Students Transcripts

INPUT/OUTPUT:
Students_Transcripts : data_out
transfer_credits_awarded : data_in
course_grades : data_in
```

```
course_grades_changes : data_in
comments_on_transcripts : data_in
security_access_level : data_in

BODY:
add either of <transfer_credits_awarded> or
              <course_grades> or
              <course_grades_changes> or
              <comments_on_transcripts>
    IF (<security_access_level> = OK)
       to either of <Students_Transcripts>.transfer_credits or
              <Students_Transcripts>.course_grades or
              <Students_Transcripts>.comments
```

```
P-Spec 1.7;7: Update Financial Records

NAME:
1.7;7

TITLE:
Update Financial Records

INPUT/OUTPUT:
Students_Financial_Accounts : data_out
instalment_plans : data_in
optioned_out_fee_students : data_in
student_payments : data_in
security_access_level : data_in
tuition_and_fees_rates : data_in

BODY:
IF (<security_access_level> = OK)
   IF (<student_payments>)
      FOR each student in <student_payments>
         credit <Students_Financial_Accounts> with <student_payments>
   ELSE IF (<optioned_out_fee_students>)
      add to <Students_Financial_Accounts>.opt_out_fee
   ELSE IF (<instalment_plans>)
      add to <Students_Financial_Accounts>.instal_plan
```

B.11 Level 2 - Execute Administrative Requests

Figure B.65 is the child data flow diagram of process 2 from level 0 data flow diagram. There is a process for each type of data file that is used to generate reports. Also, the

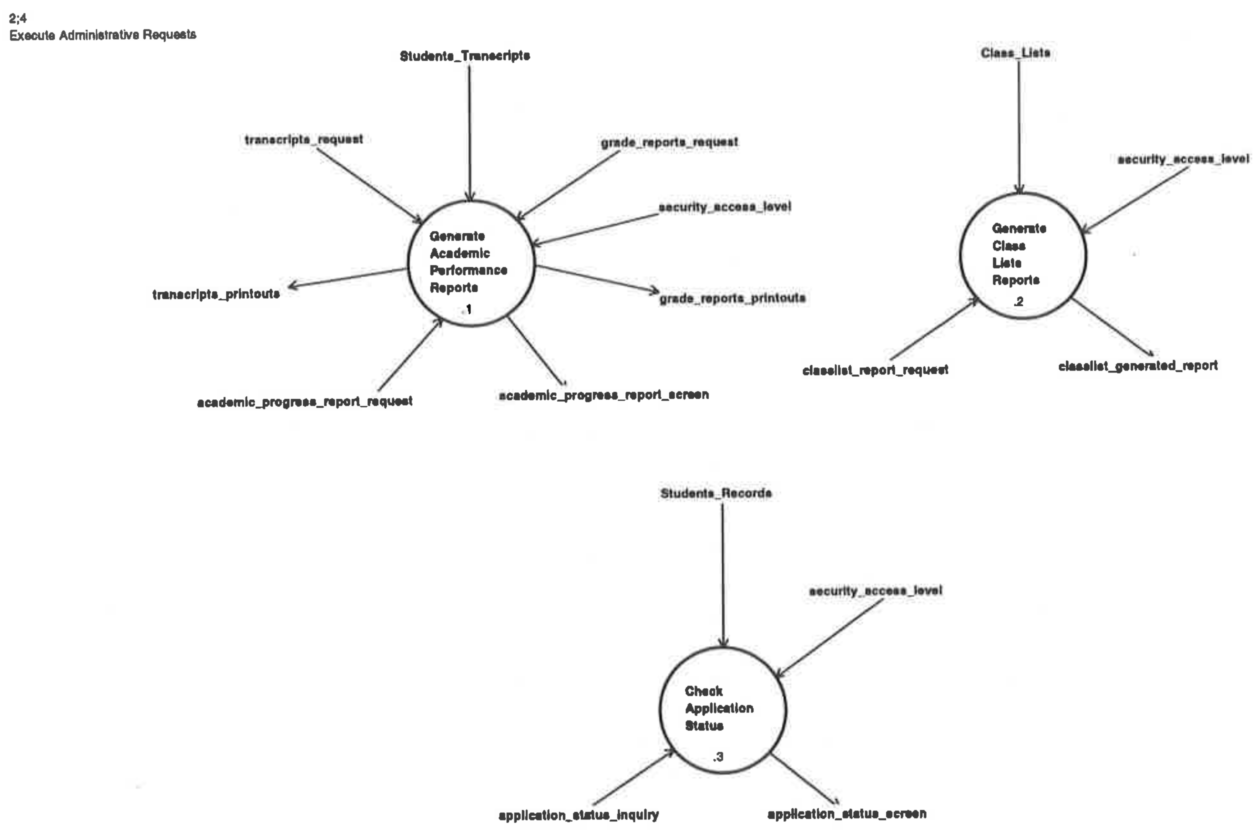

2;4
Execute Administrative Requests
Students_Transcripts
transcripts_request
grade_reports_request
security_access_level
Generate Academic Performance Reports .1
transcripts_printouts
grade_reports_printouts
academic_progress_report_request
academic_progress_report_screen
Class_Lists
security_access_level
Generate Class Lists Reports .2
classlist_report_request
classlist_generated_report
Students_Records
security_access_level
Check Application Status .3
application_status_inquiry
application_status_screen

Figure B.65: Level 2 - Execute Administrative Requests

processes are primitive processes in that they are not further decomposed. Their PSpecs broadly describe how the inputs are transformed to outputs since they are only written for the purpose of completeness in our endeavour to specify our model using Teamwork.

```
P-Spec 2.1;6: Generate Academic Performance Reports

NAME:
2.1;6

TITLE:
Generate Academic Performance Reports

INPUT/OUTPUT:
academic_progress_report_screen : data_out
grade_reports_printouts : data_out
transcripts_printouts : data_out
Students_Transcripts : data_in
transcripts_request : data_in
grade_reports_request : data_in
academic_progress_report_request : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <grade_reports_request>)
   FOR each student in <grade_reports_request>
     send <grade_reports_printouts> for session needed to printer
          from <Students_Transcripts>
ELSE IF (<security_access_level> = OK AND <transcripts_request>)
   FOR each student in <transcripts_request>
     send <transcripts_printouts> for whole academic career at
          the University from <Students_Transcripts>
ELSE IF (<security_access_level> = OK AND
                 <academic_progress_report_request>)
   show <Students_Transcripts> for student on computer terminal
        as <academic_progress_report_screen>
```

```
P-Spec 2.2;6: Generate Class Lists Reports

NAME:
2.2;6

TITLE:
Generate Class Lists Reports

INPUT/OUTPUT:
classlist_generated_report : data_out
```

```
Class_Lists : data_in
classlist_report_request : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <classlist_report_request>)
   FOR each section for which a class list is requested
      send section.<Class_Lists> as <classlist_generated_report>
         to the printer
```

```
P-Spec 2.3;7: Check Application Status

NAME:
2.3;7

TITLE:
Check Application Status

INPUT/OUTPUT:
application_status_screen : data_out
Students_Records : data_in
application_status_inquiry : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <application_status_inquiry>)
   bring up <Students_Records.status> for student requested
      as <application_status_screen>
```

B.12 Level 3 - Execute Registration Session

Figure B.66 is the child data flow diagram of process 3 in level 0 data flow diagram. Since the registration process is the emphasis of this document, this data flow diagram will be further decomposed. The State Transition Diagram in Figure B.68 depicts the three states in the registration process and how a user moves from one state to another. It is to be read in conjunction with the data flow diagram in Figure B.66.

The registration process itself starts with the user providing her student identification. That piece of information is validated and the student's record is checked for financial holds and disciplinary blocks as well as for an outstanding fee balance. These tasks are performed by the first three processes. The decision table in Figure B.67 shows the resulting outcome of the three checks from these processes. If the final outcome is positive, then the student moves from the 'Logging In' state to the 'Preprocessing' state where the student's eligibility to register and her access date are checked upon receiving

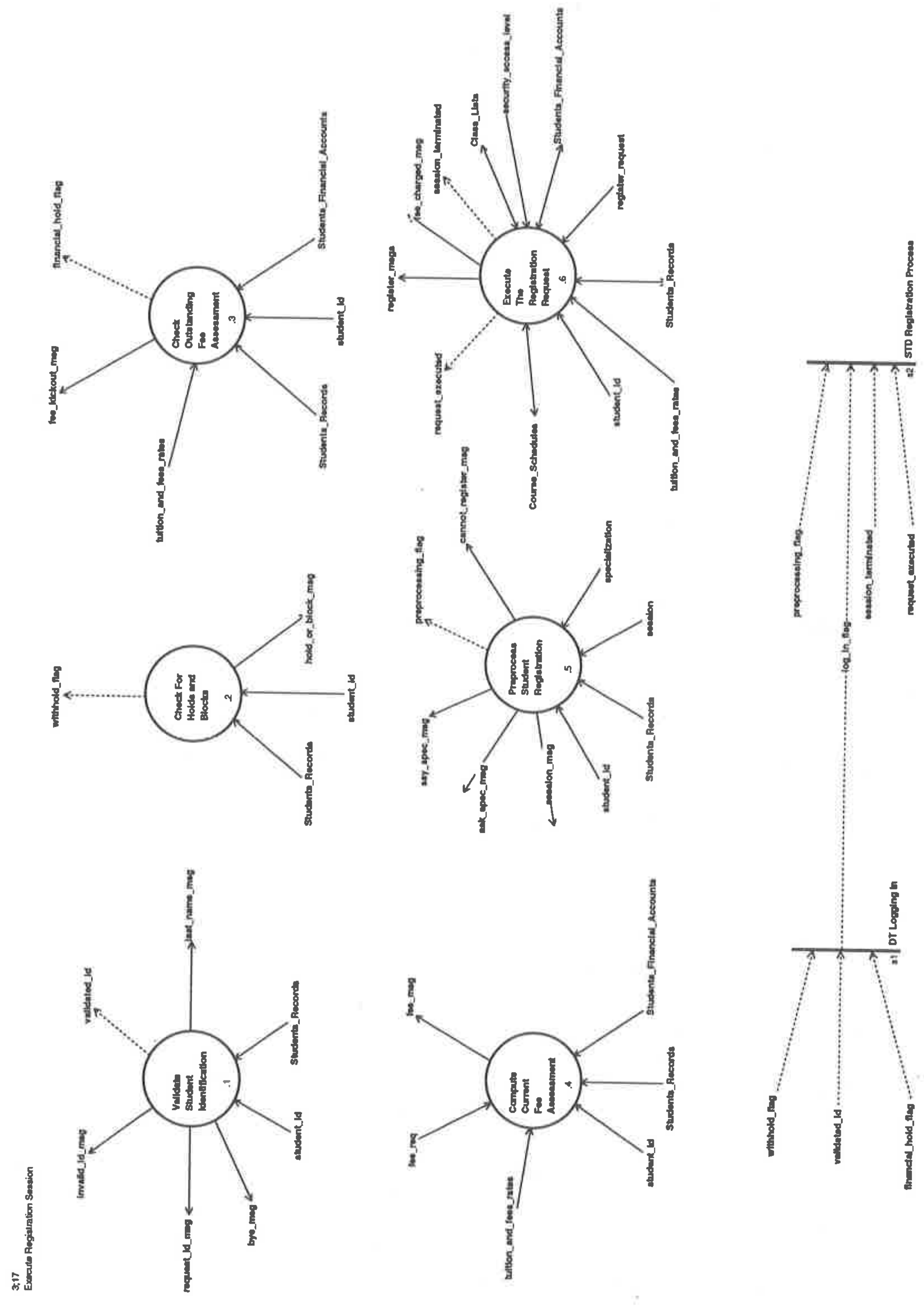


Figure B.66: Level 3 - Execute Registration Session

3-s1;5
DT Logging In

financial_hold_flag	withhold_flag	validated_id	log_in_flag
"FALSE"	"FALSE"	"TRUE"	"TRUE"
			"FALSE"

Figure B.67: DT Logging In

the session for which registration is sought (process 5). At this state, the student can request for her fee balance (process 4). If the student successfully passes the current state, then she can move to the 'Processing' state where she can request for multiple registration transactions (process 6). This state is exited when the registration process is over.

```
P-Spec 3.1;8: Validate Student Identification

NAME:
3.1;8

TITLE:
Validate Student Identification

INPUT/OUTPUT:
bye_msg : data_out
validated_id : control_out
invalid_id_msg : data_out
request_id_msg : data_out
last_name_msg : data_out
student_id : data_in
Students_Records : data_in

BODY:
attempt + 1
<validated_id> = "FALSE"
IF (<student_id format> = INVALID AND attempt < 3)
    give <request_id_msg>
ELSE IF (attempt >= 3)
    give <bye_msg>
```

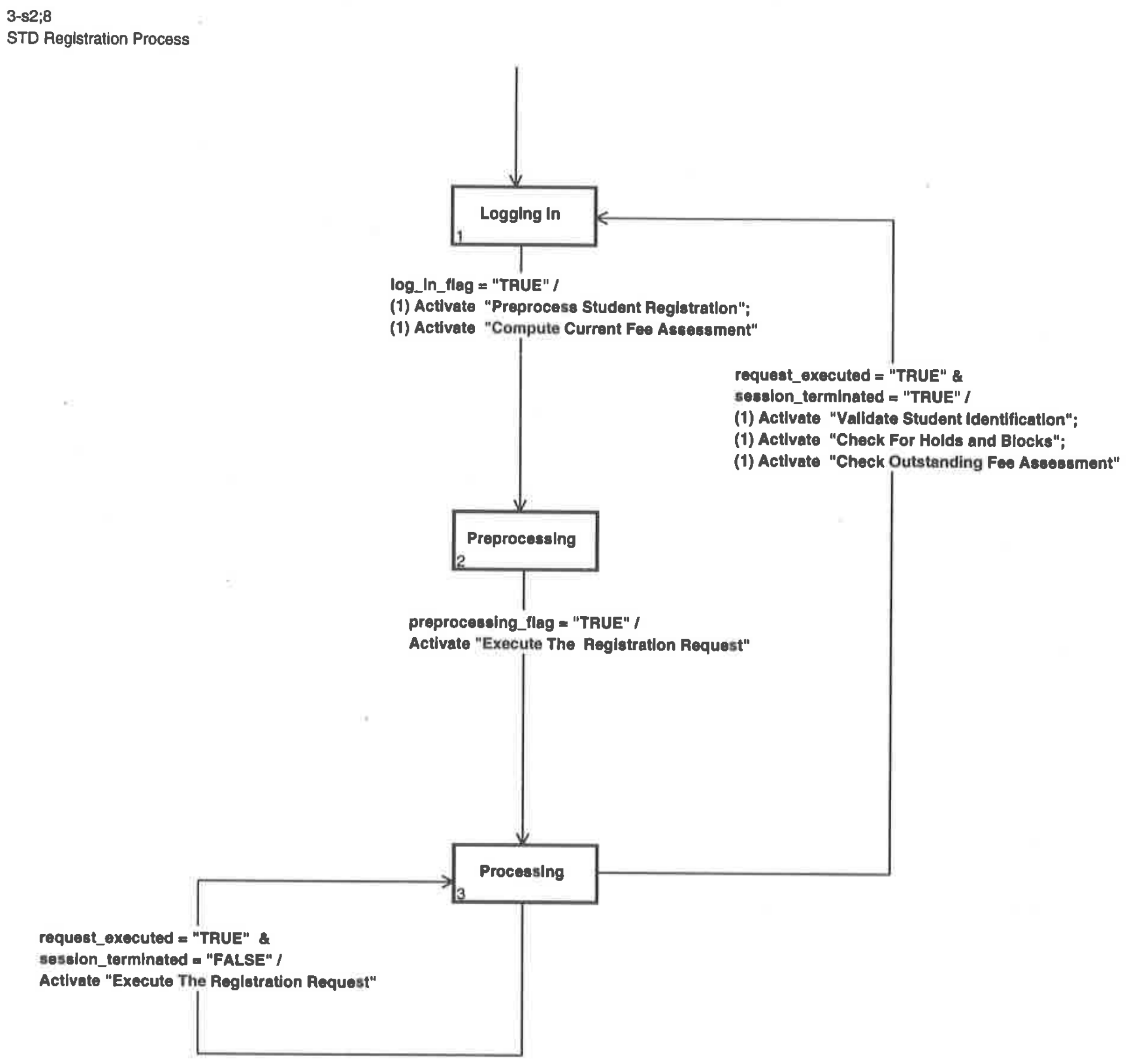

3-s2;8
STD Registration Process
Logging In
1
log_in_flag = "TRUE" /
(1) Activate "Preprocess Student Registration";
(1) Activate "Compute Current Fee Assessment"
Preprocessing
2
preprocessing_flag = "TRUE" /
Activate "Execute The Registration Request"
Processing
3
request_executed = "TRUE" &
session_terminated = "TRUE" /
(1) Activate "Validate Student Identification";
(1) Activate "Check For Holds and Blocks";
(1) Activate "Check Outstanding Fee Assessment"
request_executed = "TRUE" &
session_terminated = "FALSE" /
Activate "Execute The Registration Request"

Figure B.68: STD Registration Process

```
ELSE IF (attempt < 3)
    search <student_id> against <Students_Records>
    IF (student_number OR birth_date = INVALID)
       give <invalid_id_msg>
       give <request_id_msg>
    ELSE
          IF (found)
          give <last_name_msg> from <student_id>.<Students_Records>
          <validated_id> = "TRUE"
       ELSE
          give <invalid_id_msg>
          give <request_id_msg>
```

P-Spec 3.2;8: Check For Holds and Blocks

NAME:
3.2;8

TITLE:
Check For Holds and Blocks

INPUT/OUTPUT:
hold_or_block_msg : data_out
withhold_flag : control_out
Students_Records : data_in
student_id : data_in

BODY:

```
IF (<student_id>.<Students_Records>.hold
OR <student_id>.<Students_Records>.block = TRUE)
   give <hold_or_block_msg>
   <withhold_flag> = "TRUE"
ELSE
   <withhold_flag> = "FALSE"
```

P-Spec 3.3;8: Check Outstanding Fee Assessment

NAME:
3.3;8

TITLE:
Check Outstanding Fee Assessment

```
INPUT/OUTPUT:
financial_hold_flag : control_out
fee_kickout_msg : data_out
student_id : data_in
Students_Records : data_in
Students_Financial_Accounts : data_in
tuition_and_fees_rates : data_in

BODY:
IF (<student_id>.<Students_Records>.<Students_Financial_Accounts>
     has outstanding fee)
   give <fee_kickout_msg>
   <financial_hold_flag> = "TRUE"
ELSE
   <financial_hold_flag> = "FALSE"
```

```
P-Spec 3.4;7: Compute Current Fee Assessment

NAME:
3.4;7

TITLE:
Compute Current Fee Assessment

INPUT/OUTPUT:
fee_msg : data_out
Students_Records : data_in
Students_Financial_Accounts : data_in
student_id : data_in
fee_req : data_in
tuition_and_fees_rates : data_in

BODY:
EQUIVALENT TO 3.6.4.9 except for data_in request_executed and
   session_terminated.
```

```
P-Spec 3.5;7: Preprocess Student Registration

NAME:
3.5;7

TITLE:
Preprocess Student Registration
```

```
INPUT/OUTPUT:
session_msg : data_out
cannot_register_msg : data_out
ask_spec_msg : data_out
preprocessing_flag : control_out
say_spec_msg : data_out
Students_Records : data_in
session : data_in
student_id : data_in
specialization : data_in

BODY:
IF (<session>)
   IF (<student_id>.<Students_Records>.eligibility = FALSE OR
            <student_id>.<Students_Records>.access_date = Not OK)
       give <cannot_register_msg>
       <preprocessing_flag> = "FALSE"
   ELSE
       give <session_msg>
       IF (<student_id>.<Students_Records>.specialization = missing)
          give <ask_spec_msg>
          IF (<specialization>)
             <student_id>.<Students_Records>.specialization =
                <specialization>
             give <say_spec_msg>
       ELSE
          give <say_spec_msg> from
               <student_id>.<Students_Records>.specialization
       <preprocessing_flag> = "TRUE"
```

B.13 Level 3.6 - Execute The Registration Request

The data flow diagram in FigureB.69 is the child of process 6 in Figure B.66. The motivation for the decomposition of processes in this data flow diagram is tied to the functions to be carried out at this level.

Upon receiving a registration request, the system validates the request in process 1. If the request might result in a probable change in the data store or might require accessing a data base, it is sent to processes 2 and 4. The former process charges a transaction fee if the student has used all her free transactions. The latter process will actually perform the execution of the request. If the request does not affect a data store in any way, it is processed in bubble 3. The Process Activation Table in Figure B.70 shows which processes are activated depending on the type of request.

P-Spec 3.6.1;6: Check Validity of Request

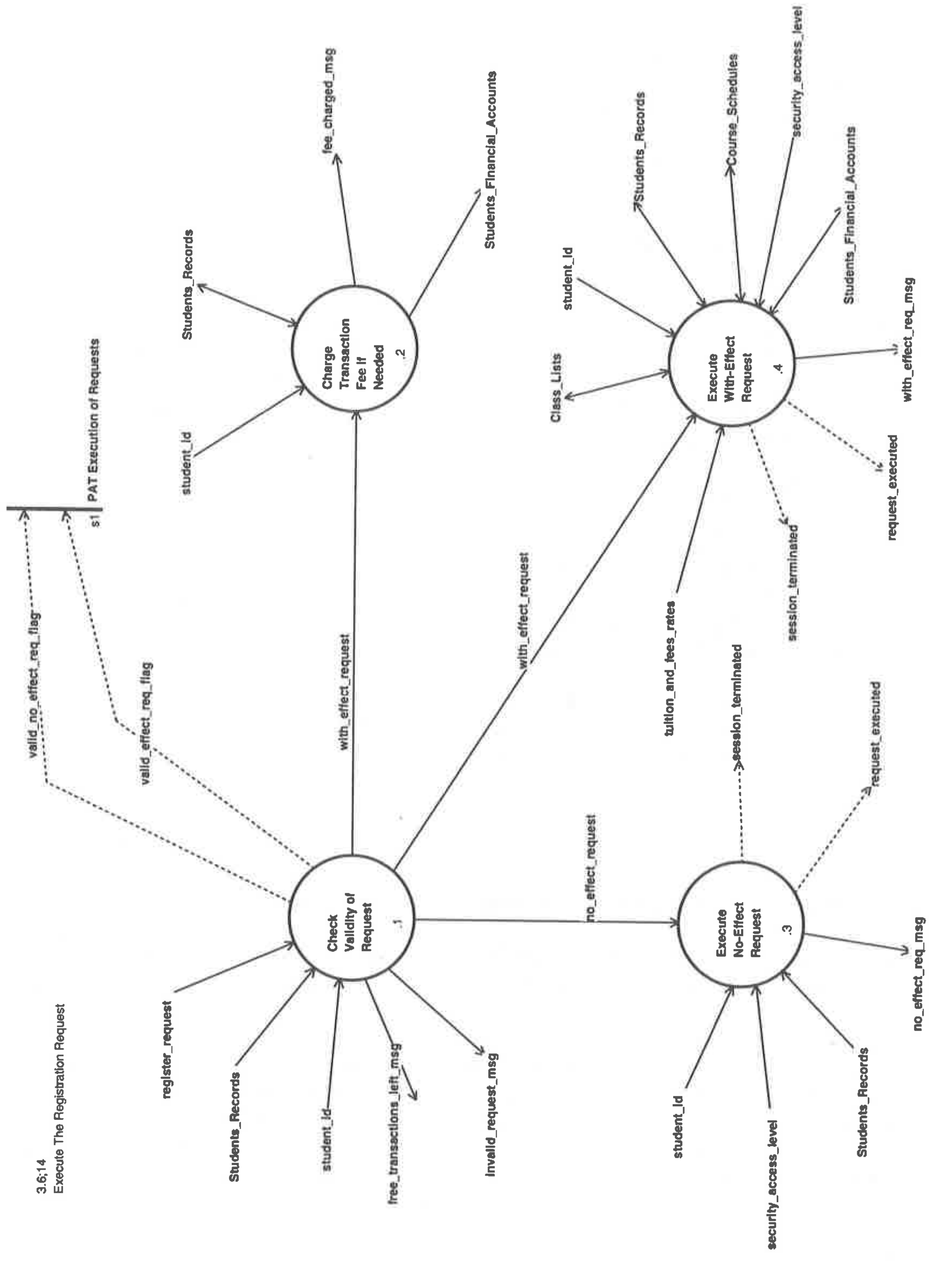


Figure B.69: Level 3.6 - Execute The Registration Request

3.6-s1;2
PAT Execution of Requests

valid_effect_req_flag	valid_no_effect_req_flag	"Charge Transaction Fee If Needed"	"Execute No-Effect Request"	"Execute With-Effect Request"
"TRUE"		1		1
	"TRUE"		1	

Figure B.70: PAT Execution Of Requests

```
NAME:
3.6.1;6

TITLE:
Check Validity of Request

INPUT/OUTPUT:
valid_effect_req_flag : control_out
valid_no_effect_req_flag : control_out
no_effect_request : data_out
invalid_request_msg : data_out
free_transactions_left_msg : data_out
with_effect_request : data_out
register_request : data_in
Students_Records : data_in
student_id : data_in

BODY:
<valid_effect_req_flag> = <valid_no_effect_req_flag> = "FALSE"
IF (<register_request>)
   IF (concurrent update of student by another user)
       give <invalid_request_msg>
   ELSE
      IF (timeout occurred before request made AND
                                maximum timeouts exceeded)
         give <bye_msg>
      IF (timeout occurred before request made OR
                                     request format = INVALID)
         give <invalid_request_msg>
      ELSE
         IF(<student_id>.<Students_Records>.schedule.<free_transactions_left>
                    <= 10)
         give <free_transactions_left_msg>
      SWITCH (<register_request>)
         CASE exit_req OR repeat_req:
                                       <no_effect_request> = <register_request>
                                       <valid_no_effect_req_flag> = "TRUE"
         CASE add_course_req OR
              confirm_req    OR
              change_req     OR
              drop_req       OR
              lookup_req     OR
              list_req       OR
              cancel_req     OR
              fee_req       OR
              add_spec_req:            <with_effect_request> = <register_request>
                                       <valid_effect_req_flag> = "TRUE"
         DEFAULT:                      give <invalid_request_msg>
```

P-Spec 3.6.2;8: Charge Transaction Fee if Needed

NAME:
3.6.2;8

TITLE:
Charge Transaction Fee if Needed

INPUT/OUTPUT:
Students_Records : data_inout
Students_Financial_Accounts : data_out
fee_charged_msg : data_out
with_effect_request : data_in
student_id : data_in

BODY:
```
IF (<student_id>.<Students_Records>.schedule.<free_transactions_left>
               <= 0)
   IF <(with_effect_request> = add_course_req OR
             drop_req OR change_req OR lookup_req)
      add transaction fee to
         <student_id>.<Students_Records>.<Students_Financial_Accounts>
      give <fee_charged_msg>
```

P-Spec 3.6.3;10: Execute No-Effect Request

```
NAME:
3.6.3;10

TITLE:
Execute No-Effect Request

INPUT/OUTPUT:
no_effect_req_msg : data_out
session_terminated : control_out
request_executed : control_out
Students_Records : data_in
student_id : data_in
no_effect_request : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK)
   IF (<no_effect_request> = repeat_req)
      give <repeat_msg> for <student_id>.<Students_Records>
      <session_terminated> = "FALSE"
   ELSE IF (<no_effect_request> = exit_req)
      activate "Compute Current Fee Assessment" for
                  <student_id>.<Students_Records>
      give <exit_msg>
      <session_terminated> = "TRUE"
   <request_executed> = "TRUE"
```

B.14 Level 3.6.4 - Execute With-Effect Request

The last data flow diagram of this model (Figure B.71) shows the execution of registration requests that access data files. At any one time, only one of processes 2 to 10 is activated depending on the request. The Process Activation Table in Figure B.72 shows how the selection is made. PSpecs 3.6.4.1 through 3.6.4.10 describe how the processing in these ten primitives processes takes place.

P-Spec 3.6.4.1;8: Determine Kind of With-Effect Request

```
NAME:
3.6.4.1;8

TITLE:
Determine Kind of With-Effect Request

INPUT/OUTPUT:
add_spec_flag : control_out
list_flag : control_out
```

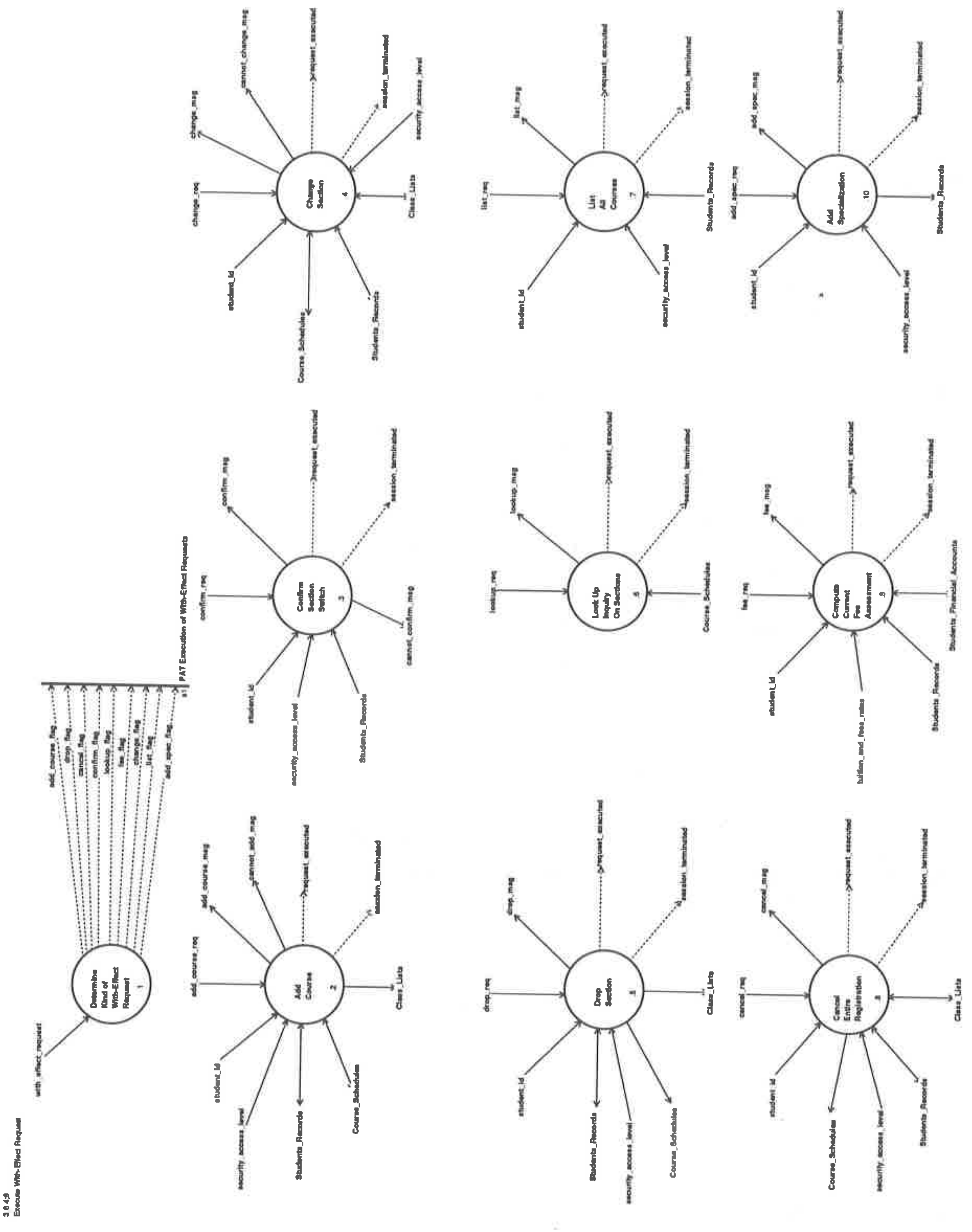


Figure B.71: Level 3.6.4 - Execute With-Effect Request

3.6.4-s1;1
PAT Execution of With-Effect Requests

add_spec_flag	list_flag	change_flag	fee_flag	add_course_flag	drop_flag	cancel_flag	confirm_flag	lookup_flag	"Add Course"	"Confirm Section Switch"	"Change Section"	"Drop Section"	"Look Up Inquiry On Sections"	"List All Courses"	"Cancel Entire Registration"	"Compute Current Fee Assessment"	"Add Specialization"
'TRUE'																	1
	'TRUE'													1			
		'TRUE'									1						
			'TRUE'													1	
				'TRUE'					1								
					'TRUE'							1					
						'TRUE'									1		
							'TRUE'			1							
								'TRUE'					1				

Figure B.72: PAT Execution of With-Effect Requests

```
change_flag : control_out
fee_flag : control_out
add_course_flag : control_out
drop_flag : control_out
cancel_flag : control_out
confirm_flag : control_out
lookup_flag : control_out
with_effect_request : data_in

BODY:
<add_spec_flag> = <list_flag> = <change_flag> = <fee_flag> =
<add_course_flag> = <drop_flag> = <cancel_flag> = <confirm_flag> =
<lookup_flag> = "FALSE"

SWITCH (<with_effect_request>)
   CASE add_spec_req:     <add_spec_flag> = "TRUE"
   CASE list_req:         <list_flag> = "TRUE"
   CASE change_req:       <change_flag> = "TRUE"
   CASE fee_req:          <fee_flag> = "TRUE"
   CASE add_course_req:   <add_course_flag> = "TRUE"
   CASE drop_req:         <drop_flag> = "TRUE"
   CASE cancel_req:       <cancel_flag> = "TRUE"
   CASE confirm_req:      <confirm_flag> = "TRUE"
   CASE lookup_req:       <lookup_flag> = "TRUE"
```

P-Spec 3.6.4.2;9: Add Course

```
NAME:
3.6.4.2;9

TITLE:
Add Course

INPUT/OUTPUT:
Course_Schedules : data_inout
Students_Records : data_inout
cannot_add_msg : data_out
add_course_msg : data_out
session_terminated : control_out
request_executed : control_out
Class_Lists : data_out
add_course_req : data_in
student_id : data_in
security_access_level : data_in

BODY:
```

```
IF (<security_access_level> = OK AND <add_course_req>)
   IF (<student_id>.<Students_Records>.STT = OK)
      give <cannot_add_msg>
   ELSE
      IF (course.<Course_Schedules>.openings > 0 AND
          <student_id>.<Students_Records>.max_credits not exceeded)
         add course to <student_id>.<Students_Records>.schedule
         add <student_id>.<Students_Records> to <Class_Lists> for
              that course
         course.<Course_Schedules>.openings - 1
         <student_id>.<Students_Records>.schedule.free_transactions_left
              - 1
         give <add_course_msg>
      ELSE
         give <cannot_add_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.3;7: Confirm Section Switch

```
NAME:
3.6.4.3;7

TITLE:
Confirm Section Switch

INPUT/OUTPUT:
cannot_confirm_msg : data_out
session_terminated : control_out
request_executed : control_out
confirm_msg : data_out
confirm_req : data_in
student_id : data_in
Students_Records : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <confirm_req>)
   IF (<student_id>.<Students_Records>.STT = OK)
      give <cannot_confirm_msg>
   ELSE
      search in <student_id>.<Students_Records>.schedule for
           course to be confirmed
      IF (found)
         give <confirm_msg>
      ELSE
```

```
         give <cannot_confirm_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.4;7: Change Section

NAME:
3.6.4.4;7

TITLE:
Change Section

INPUT/OUTPUT:

```
Class_Lists : data_inout
Students_Records : data_inout
Course_Schedules : data_inout
session_terminated : control_out
request_executed : control_out
cannot_change_msg : data_out
change_msg : data_out
change_req : data_in
student_id : data_in
security_access_level : data_in
```

BODY:

```
IF (<security_access_level> = OK AND <change_req>)
   IF (<student_id>.<Students_Records>.STT = OK)
      give <cannot_change_msg>
   ELSE
      IF (new_section.<Course_Schedules>.openings > 0)
         search in <student_id>.<Students_Records>.schedule
              for old_section
         IF (found)
            change old_section to new_section in
               <student_id>.<Students_Records>.schedule
            remove <student_id>.<Students_Records> from
                <Class_Lists>.old_section
            add <student_id>.<Students_Records> to
                <Class_Lists>.new_section
            new_section.<Course_Schedules>.openings - 1
            old_section.<Course_Schedules>.openings + 1
            <student_id>.<Students_Records>.schedule.free_transactions_left
                 - 1
            give <change_msg>
         ELSE
            give <cannot_change_msg>
      ELSE
```

```
         give <cannot_change_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.5;7: Drop Section

```
NAME:
3.6.4.5;7

TITLE:
Drop Section

INPUT/OUTPUT:
Students_Records : data_inout
Course_Schedules : data_out
session_terminated : control_out
request_executed : control_out
Class_Lists : data_out
drop_msg : data_out
drop_req : data_in
student_id : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <drop_req>)
   IF (<student_id>.<Students_Records>.STT = OK)
      give <cannot_drop_msg>
   ELSE
      search for section in <student_id>.<Students_Records>.schedule
      IF (found)
         remove section from <student_id>.<Students_Records>.schedule
         remove <student_id.<Students_Records> from
                  <Class_Lists>.section
         section.<Course_Schedules>.openings + 1
         <student_id>.<Students_Records>.schedule.free_transactions_left
                - 1
         give <drop_msg>
      ELSE
         give <cannot_drop_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.6;6: Look Up Inquiry On Sections

```
NAME:
3.6.4.6;6

TITLE:
Look Up Inquiry On Sections

INPUT/OUTPUT:
session_terminated : control_out
request_executed : control_out
lookup_msg : data_out
lookup_req : data_in
Course_Schedules : data_in

BODY:
IF (<lookup_req>)
   look up the value of section.<Course_Schedules>.openings
   student_id.<Students_Records>.schedule.free_transactions_left
          - 1
   give <lookup_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.7;7: List All Courses

```
NAME:
3.6.4.7;7

TITLE:
List All Courses

INPUT/OUTPUT:
session_terminated : control_out
request_executed : control_out
list_msg : data_out
list_req : data_in
Students_Records : data_in
student_id : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <list_req>)
   go to <student_id>.<Students_Records>.schedule
   give <list_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

```
P-Spec 3.6.4.8;6: Cancel Entire Registration

NAME:
3.6.4.8;6

TITLE:
Cancel Entire Registration

INPUT/OUTPUT:
Students_Records : data_inout
Class_Lists : data_inout
Course_Schedules : data_out
session_terminated : control_out
request_executed : control_out
cancel_msg : data_out
cancel_req : data_in
student_id : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <cancel_req>)
    FOR each section in <student_id>.<Students_Records>.schedule
            for the session
       look up in <Class_Lists>.section.<student_id> and remove
            the student
       delete section from <student_id>.<Students_Records>.schedule
       section.<Course_Schedules>.openings + 1
    give <cancel_msg>
    <request_executed> = "TRUE"
    <session_terminated> = "FALSE"
```

```
P-Spec 3.6.4.9;8: Compute Current Fee Assessment

NAME:
3.6.4.9;8

TITLE:
Compute Current Fee Assessment

INPUT/OUTPUT:
Students_Financial_Accounts : data_inout
session_terminated : control_out
```

```
request_executed : control_out
fee_msg : data_out
fee_req : data_in
student_id : data_in
Students_Records : data_in
tuition_and_fees_rates : data_in

BODY:
IF (<fee_req>)
   calculate new fees due based on the courses and credits registered in
       <student_id>.<Students_Financial_Accounts>
   put new fee and due dates in
          <student_id>.<Students_Financial_Accounts>
   give <fee_msg> from <student_id>.<Students_Financial_Accounts>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

P-Spec 3.6.4.10;6: Add Specialization

```
NAME:
3.6.4.10;6

TITLE:
Add Specialization

INPUT/OUTPUT:
session_terminated : control_out
request_executed : control_out
Students_Records : data_out
add_spec_msg : data_out
add_spec_req : data_in
student_id : data_in
security_access_level : data_in

BODY:
IF (<security_access_level> = OK AND <add_spec_req>)
   add spec to <student_id>.<Students_Records>
   give <add_spec_msg>
   <request_executed> = "TRUE"
   <session_terminated> = "FALSE"
```

B.15 Data Dictionary Entries

```
academic_progress_report_request (data flow, pel) =
    * This is a request made by faculty advisers when they meet with
      students to advise them on courses to take in order to fulfill
      their degree requirements.  Faculty advisers want to have a
      list of courses already completed by the student and the grades
      earned in those courses. *


academic_progress_report_screen (data flow, pel) =
    * This is information that is made available on a computer screen
      to faculty advisers when they meet with students.  This
      information contains a list of the courses that the student
      has completed and the grades that she has earned in them.  It
      also lists the required courses that the student needs
      to complete and the possible electives to fulfill her degree
      requirements. *


access_dates (data flow, pel) =
    * This consists of a table of dates at which time students can
      access the Telereg telephone registration system and the
      criteria the students must fulfill in order to be eligible
      to register at those days.  The access dates are computed
      according to a formula that takes into account
      criteria such as year, degree program, category,
      previous session average or admission average.  Students
      with the highest averages have the earliest access dates for
      their year level. Unclassified students are the last ones
      eligible to register. *


add_course_flag (control flow, del) =
    [ "TRUE"
     | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to add a course to the student's course schedule for
      a particular session. *


add_course_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to add a course to a student's course schedule for a
      particular session to confirm that the request has been executed. *


add_course_req (data flow, pel) =
    * This request consists of a command to add a course and the
      course number to be added. *


add_spec_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to add a specialization or a degree program to the
      student's record. *


add_spec_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to add a specialization or degree program to the
      student's record to confirm that the request has been executed.
      In the case of a student registering through the telephone
      registration system, this feedback is a voice response whereas
      in the case of the Registrar's office or a department
      administrator registering on behalf of a student this feedback
      is given in the form of a print message on the computer
      terminal.  While the form of delivery might be different, the
      content of the message is the same. *


add_spec_req (data flow, pel) =
    * This request consists of a command to add a specialization
      or degree program and the specialization or degree program
      to be added. *


adm_database_info (data flow) =
    [ new_students_biblio_info
      | newly_admitted_students
      | new_faculty_biblio_info
      | transfer_credits_awarded
    ]
    * This is the list of information that the department provides
      to the system in order to update its databases. *


adm_report_request (data flow, alias) =
    application_status_inquiry
    * The kind of report request that the Admissions Office can ask
      for. *


adm_report_screen (data flow, alias) =
    application_status_screen
    * The kind of report screen that can be displayed at the request of
      the Admissions Office. *


administrative_info (data flow) =
    [ course_grades
      | course_grades_changes
      | comments_on_transcripts
      | updated_student_biblio_info
      | updated_faculty_biblio_info
      | course_room_assignments
      | classroom_bookings
      | course_catalog_info
      | course_catalog_info_changes
    ]
    *  These are information used in one form or the other to
       maintain data files that are used in activities other than
       registration at the Registrar's office. *


application_status_inquiry (data flow, pel) =
    * This is an inquiry that can be made to the Student
      Information System about the admission status of undergraduate
      applicants not yet admitted to the University. *


application_status_screen (data flow, pel) =
    * This is a screen that is displayed after an
      application_status_inquiry to the Student Information System
      has been submitted to query about the admissions status of an
      undergraduate applicant. *


ask_spec_msg (data flow, pel) =
    * This is a message to prompt the student or the Registrar
      or Department administrator registering on behalf of the student
      for a specialization or degree program if either one is found
      missing in the student's record. *


birth_date (data flow, pel) =
    * It is also referred to as a pin number. *


bye_msg (data flow, pel) =
    * This is a message that is given by the system if the student
      fails to enter a valid student identification within three
      attempts. The system will not allow the student to proceed. *
```

```
cancel_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is
      a request to cancel the entire registration of that student
      for the particular session. *


cancel_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to cancel the student's entire registration for a
      particular session to confirm that the request has been executed.  *


cancel_req (data flow, pel) =
    * This request consists of a command to cancel the student's entire
      registration for a particular session. *


cannot_add_msg (data flow, pel) =
    * This is a message that is given when the system fails to add the
      requested course to the student's course schedule because of many
      possible reasons: the class limit for the course has been reached,
      the requested course is an invalid one, the student has already
      registered for the course, and so on. *


cannot_change_msg (data flow, pel) =
    * This is a message that is given when the system is not able
      to change a section for a course as requested by the student.
      This might be because that section is full, the specified
      section is an invalid one, and so on. *


cannot_confirm_msg (data flow, pel) =
    * This is a message that is given when the system is not able to
      confirm a section switch for a course as requested by the
      student.  This might be because the student has not previously
      registered in that section or the student has a Standard
      TimeTable which implies that she cannot change her schedule,
      and so on. *


cannot_register_msg (data flow, pel) =
    * This is a message that is given if the student is attempting to
      register before the time specified by her access date
      or if the student is not eligible to register for courses in
      the session specified in the case of a low grade average. *


cash_database_info (data flow) =
    [ optioned_out_fee_students
      | instalment_plans
      | student_payments
    ]
    * This is the kinds of information that the Cashier's Office
      provides to the system in order to update its databases. *


change_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to change a section for a course in the student's
      course schedule for a particular session. *


change_msg (data flow, pel) =
    * This is the feedback that is given by the system upon
      executing a request to change a course section in the student's
      course schedule to confirm that the request has been executed. *


change_req (data flow, pel) =
    * This request consists of a command to change a course section and
      the section to change to. *


Class_Lists (store) =
    * This database consists of the lists of students who are enrolled
      in each of the courses offered by the University. *


classlist_generated_report (data flow, pel) =
    * This is the report that is generated for the department that
      has requested for class lists of all the students that are
      enrolled in the courses offered by the department. *


classlist_report_request (data flow, pel) =
    * This is a request for class lists of all the students that have
      enrolled in all the courses that are offered by the requesting
      department. *
```

```
classroom_bookings (data flow, pel) =
    * This is a request to book a room in a particular building at a
      specified time. *


comments_on_transcripts (data flow, pel) =
    * These are pieces of information that are included in
      students' transcripts e.g. when they are expected to
      graduate, whether they are on the honor roll and so on. *


confirm_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to confirm a section switch to the student's course
      schedule for a particular session. *


confirm_msg (data flow, pel) =
    * This is the feedback that is given by the system upon
      executing a request to confirm that a section switch has
      been made. *


confirm_req (data flow, pel) =
    * This request consists of a command to confirm that a section
      switch has been made and the course section to be checked. *


Course_Catalog (store) =
    * This file consists of descriptions of courses offered by
      the University. *


course_catalog_info (data flow, pel) =
    * This consists of descriptions of courses offered by the various
      departments in the University. *


course_catalog_info_changes (data flow, pel) =
    * This consists of changes that are made to the descriptions
      of courses offered by the University. *


course_grades (data flow, pel) =
    * This is a list of students who have enrolled in a course
```

```
      and the grades they have obtained in the course as well as
      the course number. *


course_grades_changes (data flow, pel) =
    * This consists of the changes in grades that course instructors
      might have seen fit to make after the course grades have been
      sent to the Registrar's office. *


course_room_assignments (data flow, pel) =
    * This consists of a list of the rooms that have been assigned
      for each of the courses (lectures, labs and tutorials) that are
      being offerred by the University.  It also includes the times
      and days for which the rooms need to be reserved. *


Course_Schedules (store) =
    * This database consists of a list of all the courses that are
      being offered by all the departments in the University for a
      particular session, the days and times at which they are
      offered, the meeting place and the course instructor. *


create_new_session_request (data flow, pel) =
    * This is a request that is made by the Registrar's Office in
      order to initiate the course scheduling process. Through
      the roll-over process, this request will create eligibilities
      for continuing students to register in the session created. *


dept_database_info (data flow, alias) =
    dept_pgm_promotions
    * This is a list of information that the department provides to the
      system in order to update its databases. *


dept_pgm_promotions (data flow, pel) =
    * Departments can indicate whether they have approved students to
      their programs based on their academic performance. *


dept_register_requests (data flow, alias) =
    registration_requests
    * A department administrator registering on behalf of a student
      registers through the online registration system via a computer
      terminal rather than through the Telereg telephone registration
      system.  While the former form of interaction with the system
      differs from the latter, the content of the requests is the same. *
```

```
dept_register_screen (data flow, alias) =
    registration_feedback
    * A department administrator registering on behalf of a student
      receives feedback on the request by a print message on a
      computer terminal.  *


dept_report (data flow, alias) =
    classlist_generated_report
    * The kind of report that can be generated at the request of any
      department in the University. *


dept_report_request (data flow, alias) =
    classlist_report_request .
    * The kind of report that any department at the University can
      ask for. *


disciplinary_blocks (data flow, pel) =
    * These are restrictions that are placed on a student's record
      to prevent her from registering.  These restrictions arise
      because of academic reasons such as failure to keep in good
      academic standing and so on. *


drop_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to drop a course section from the student's course
      schedule for a particular session. *


drop_msg (data flow, pel) =
    * This is the feedback that is given by the system upon
      executing a request to drop a course section from a student's
      course schedule to confirm that the request has been executed. *


drop_req (data flow, pel) =
    * This request consists of a command to drop a course section and
      the course section to be dropped. *


eligibility_rejection (data flow, pel) =
    * This piece of information is provided by the Registrar's Office
      to indicate the list of students who will be denied the
      eligibility to register for whatever reasons deemed appropriate. *


exit_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing a
      request to exit the registration session. *


exit_req (data flow, pel) =
    * This request consists of a command to exit the registration
      process. *


Faculty_Records (store) =
    * This database consists of bibliographical information and
      other related information on the faculty. This information is
      kept on file by the Registrar's Office. *


faculty_switch (data flow, pel) =
    * This information provided by the Registrar's Office consists
      of the changes of instructors who will be teaching certain
      courses offered in a particular session. *

fee_charged_msg (data flow, pel) =
    * This message is given out to indicate to the student that a
      fee was charged for the request that she has made.  A fee
      is charged for a request if the student has already used up
      the maximum number of free transaction requests that she
      is allowed for each session. *


fee_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is a
      request to inquire about the student's current fee assessment. *


fee_kickout_msg (data flow, pel) =
    * This message is given by the system if it finds that the student
      who is attempting to register has an outstanding financial
      account.  The student will not be able to register. *


fee_msg (data flow, pel) =
    * This is the feedback given by the system upon executing
      a request to inquire about a student's current fee assessment.
      The dollar amount owed and the dates at which they are due are
      given. *


fee_req (data flow, pel) =
    * This request consists of a command to inquire about a particular
      student's current fee assessment. *


financial_hold_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is used to indicate that a financial hold has been
      placed on a student's record because she has an
      outstanding financial account with the University.  This flag
      is set to "FALSE" if there are no outstanding account and "TRUE"
      otherwise. *


financial_holds (data flow, pel) =
    * These are restrictions that are placed on a student's record
      to prevent her from registering.  These restrictions arise
      because of financial reasons such as an outstanding financial
      balance or any other reasons that would require for the student
      to clear with the financial departments first before she
      is allowed to register. *


free_transactions_left_msg (data flow, pel) =
    * If a student has used her free transactions requests
      so that there are only ten or less for which she might
      be eligible for in a session, then this message will warn
      her of the number of free transactions left. *


grade_reports_printouts (data flow, pel) =
    * These are printouts of statements of grades for students
      enrolled in a session. *


grade_reports_request (data flow, pel) =
    * This is a request to generate the grade reports of students
      enrolled in a session. *


hold_or_block_msg (data flow, pel) =
    * This is a message that is given by the system to indicate
      that the student has a financial hold or a disciplinary block
      on her record and the appropriate party she should
      contact for more information. *


instalment_plans (data flow, pel) =
    * This is a list of students who have been approved to settle
      their tuition and fees account in instalments and the plan under
      which they have agreed to conform. *


invalid_id_msg (data flow, pel) =
    * This is the message that is given if the student_id provided
      is found to be invalid. *


invalid_request_msg (data flow, pel) =
    * This is a message that is given if the request that the student
      provided is found to be a faulty one or an invalid one or if
      the student is being concurrently updated by another user. *


last_name_msg (data flow, pel) =
    * This personalised message for each student who is trying to
      register gives the first four letters of the last name of the
      student after the student gives her student_id. *


list_flag (control flow, del) =
     [ "TRUE"
       | "FALSE"
     ]
    * This is used to indicate that the registration request is a
      request to list all the courses for which a student has
      registered in for a particular session. *


list_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to list all of the courses that a student has
      registered for in a particular session. *


list_req (data flow, pel) =
    * This request consists of a command to list all of the courses
      for which the student has registered for in a particular session. *
```

```
log_in_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration session has
      successfully completed the "Logging In" state in that the
      student_id has been validated, there is neither a financial_hold
      nor a disciplinary_block on the student's record and the attempt
      of signing on has not exceeded three times (flag set to "TRUE").*


lookup_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This is used to indicate that the registration request is an
      inquiry about available sections in a course. *


lookup_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to look up available sections for a course. *


lookup_req (data flow, pel) =
    * This request consists of a command to inquire about available
      sections in a course and the section to inquire about. *


max_credits (data flow, pel) =
    * This is the maximum number of credits that a school or faculty
      would allow a student in their program or major to register
      in a session. *


new_faculty_biblio_info (data flow, pel) =
    * This is bibliographical information for the list of new faculty
      who have been hired by the University.  This information
      includes first and last names, address, birth date, department
      by which they were hired, and so on. *


new_students_biblio_info (data flow, pel) =
    * This is bibliographical information for the list of new students
      who have been accepted to the University.  This information
      includes first and last names, address, birth date, student
      number assigned to the students by the Admissions Office, major,
      program, and so on. *
```

```
newly_admitted_students (data flow, pel) =
    * The list of applicants who have been admitted to the University. *


no_effect_req_msg (data flow) =
    [ exit_msg
      | repeat_msg
    ]
    * The kinds of messages that can be issued after a
      no_effect_request is executed. *


no_effect_request (data flow) =
    [ repeat_req
      | exit_req
    ]
    * The kinds of registration request that require little or no
      processing and that do not affect a student's record. *


optioned_out_fee_students (data flow, pel) =
    * This is a list of students who have chosen and are qualified to
      option out of certain fees.  This list contains the names of
      the students and the fees for which they have opted out of. *


pin_number_change (data flow, pel) =
    * The list of students for whom there will be a pin number change
      and the new pin numbers.*


preprocessing_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the student who is
      attempting to register is eligible to register and is doing so
      at a time in accordance with the access dates set by the
      Registrar's Office. Otherwise the flag is set to "FALSE". *


reg_database_info (data flow) =
    [ registration_info
      | administrative_info
    ]
    * This is a list of information that the Registrar's Office
      provides to the system in order to update its databases. *
```

```
reg_register_requests (data flow, alias) =
    registration_requests
    * A Registrar's Office personnel registering on behalf of a
      student registers through the online registration system or
      the Hotline via a computer terminal rather than through the
      Telereg telephone registration system. The content of the
      requests in either system is the same. *


reg_report (data flow) =
    [ grade_reports_printouts
      | transcripts_printouts
    ]
    * The kinds of report printouts that can be generated at the
      request of the Registrar's Office. *


reg_report_requests (data flow) =
    [ transcripts_request
      | grade_reports_request
    ]
    * The kinds of report requests that the Registrar's Office can
      ask for.  *


reg_visual_feedback (data flow, alias) =
    registration_feedback
    * These are the visual version of the messages that are given
      out by the system.  These messages are exactly identical to
      the ones issued by Telereg telephone registration system except
      that they are integrally displayed on a computer screen. *


register_msgs (data flow) =
    [ fee_charged_msg
      | free_transactions_left_msg
      | invalid_request_msg
      | no_effect_req_msg
      | with_effect_req_msg
      | fee_charged_msg
    ]
    * The kinds of messages that can be given by the system during a
      registration session. *


register_request (data flow) =
    with_effect_request
    + no_effect_request
    * The kinds of registration requests that can be made to the
      system during a registration session. *


registration_feedback (data flow) =
    [ request_id_msg
      | invalid_id_msg
      | bye_msg
      | last_name_msg
      | fee_kickout_msg
      | hold_or_block_msg
      | fee_msg
      | session_msg
      | say_spec_msg
      | ask_spec_msg
      | cannot_register_msg
      | register_msgs
    ]
    * These are the kinds of verbal messages that the Telereg-
      Student Information System can give out to the students. *


registration_info (data flow) =
    [ faculty_switch
      | create_new_session_request
      | session_schedules
      | updates_to_session_schedules
      | pin_number_change
      | access_dates
      | financial_holds
      | disciplinary_blocks
      | eligibility_rejection
    ]
    + tuition_and_fees_rates
    * These constitute information that are used in one form or
      the other to update data files accessed during
      registration through either the telephone registration
      system or via a computer terminal. *


registration_requests (data flow) =
    [ student_id
      | fee_req
      | specialization
      | session
      | register_request
    ]
    * The kinds of inputs and requests that a student can provide to
      the Telereg-Student Information System. These requests are in
```

```
        the form of touch tones made by pressing combinations of
        buttons on a touchtone telephone keypad. *


repeat_msg (data flow, pel) =
    * This is the feedback that is given by the system upon executing
      a request to repeat the previous message. *


repeat_req (data flow, pel) =
    * This request consists of a command to repeat the previous
      message. *


request_executed (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the registration
      request provided has been successfully executed and the system
      is ready to accept a new registration request. Otherwise, it
      is set to "FALSE". *


request_id_msg (data flow, pel) =
    * This is a message to request the user for a student_id. *


retry_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the student is
      allowed to try again on the same registration request since
      she is not on her third attempt yet. *


Room_Booking_Schedule (store) =
    * This file contains the list of all rooms available for courses
      and meetings in all the buildings of the University and the
      times at which they are booked and the parties who have booked
      them. *


say_spec_msg (data flow, pel) =
    * This message is given as a feedback to the student to indicate
      to her the specialization or degree program for which
      she is admitted to. *
```

```
sch_database_info (data flow) =
    [ STT_schedules
      | max_credits
    ]
    * This is a list of information that the various schools and
      faculties provide to the system in order to update its databases. *


sch_report_request (data flow, alias) =
    academic_progress_report_request
    * This is the kind of report request that the Schools and
      Faculties can ask for. *


sch_report_screen (data flow, alias) =
    academic_progress_report_screen
    * This is the kind of report that can be displayed on a
      computer terminal at the request of Schools and Faculties. *


security_access_level (data flow, del) =
    * Depending on who the user is, a user_id is associated with
      a security_access_level.  Various security_access_level have
      differing levels of access to the system. *


session (data flow, pel) =
    * The session for which the student wishes to register for. *


session_msg (data flow, pel) =
    * The message that is given by the system as a feedback to
      indicate the session for which the student wishes to register in. *


session_schedules (data flow, pel) =
    * This is the list of courses offered by all the departments in
      the University for a particular session, the meeting times for
      each course, and the instructor for each course. *


session_terminated (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the student has
      completed her registration session and will not provide
      any more requests. By default, the flag is set to "FALSE". *
```

```
specialization (data flow, pel) =
    * This is the degree program or specialization for which the
      student is eligible. *


STT_schedules (data flow, pel) =
    * The Standard TimeTable (STT) is a number of course sections
      that have been grouped together within a catalog number to form
      a conflict free registration package required for a specific
      program. By specifying the catalog number associated with the
      STT, students are automatically registered in each course
      section included within the STT. *


student_id (data flow, pel) =
    student_number
    + birth_date
    * The identification that a student provides in order to register. *


student_number (data flow, pel) =
    * A unique identification number that is assigned by the
      University to each student. *


student_payments (data flow, pel) =
    * This is the list of students who have made a payment towards
      their tuition and fees by either means of cash, cheque, or
      credit card and the amounts paid. *


student_register_requests (data flow, alias) =
    registration_requests
    * These requests are in the form of touch tones made by pressing
      combinations of buttons on a touchtone telephone keypad. *


Students_Financial_Accounts (store) =
    * This file contains financial records for all the students in the
      University. It contains a history of the amounts due and paid,
      the dates at which the payments are due and have been paid. *


Students_Records (store) =
    * This database contains all kinds of information pertaining to
      the students at the University.  These include: bibliographical
      information, financial holds, disciplinary blocks, access dates
      to the Telephone Registration System, eligibility to register,
      student number, maximum number of credits, course schedules for
      the sessions attended at the University, and so on. *


Students_Transcripts (store) =
    * This database consists of the list of students who have enrolled
      at the University at one point in time, the sessions and terms
      for which they were enrolled, the courses they signed up for and
      the grades earned in each of the courses.  Credits that are
      transferred in from other colleges and Universities are also
      kept in this database. *


transcripts_printouts (data flow, pel) =
    * These are the printouts of the academic transcripts of those
      students for whom printouts have been requested for by the
      Registrar's Office. *


transcripts_request (data flow, pel) =
    * This is a request to print the academic transcript of a student. *


transfer_credits_awarded (data flow, pel) =
    * This is the list of transfer credits that have been awarded to
      students who transfer in from other unversities.  This
      information is passed on from the Admissions Office to the
      Student Information System. *


tuition_and_fees_rates (data flow, pel) =
    * This is the list of tuition and fees rates for the various
      classifications of students attending the University and the
      dates at which they are effective. *


updated_faculty_biblio_info (data flow, pel) =
    * This is updated bibliographical information for the faculty
      at the University. *


updated_student_biblio_info (data flow, pel) =
    * This is updated bibliographical information for the students
      enrolled at the University. *


updates_to_session_schedules (data flow, pel) =
    * These are the updates that have been made to the list of
```

```
      courses offered by the University for a particular session,
      the meeting times and the instructor for each course. *


user_id (data flow) =
    * This is an identification that is given to the administrative
      users of the system.  Each identification is unique to a user
      and is associated with a security access level in the system. *


User_Ids_Passwords (store) =
    * This file consists of a table of user_ids and passwords and
      security_access_level.  Each user_id is associated with an
      access level which limits the kind of activities and
      information available to that user. *


valid_effect_req_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the registration
      request submitted (which is also a request that will change
      the student's record, transcript, or other database) is a valid
      one and that nobody else is trying to access the same student's
      record. Otherwise, it is set to "FALSE". *


valid_no_effect_req_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the registration
      request submitted (which is also a request that will not
      change the student's record, transcript, or other database)
      is a valid one and that nobody else is trying to access the
      same student's record. Otherwise, it is set to "FALSE". *


validated_id (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" to indicate that the student_id
      provided has a valid input format and that the student number
      and birth date are both also valid. *


vocal_feedback (data flow, alias) =
    registration_feedback
    * These messages are given by the system verbally through a phone
      receiver in response to the registration requests. *


with_effect_req_msg (data flow) =
    [ add_course_msg
      | cannot_add_msg
      | confirm_msg
      | change_msg
      | cannot_change_msg
      | drop_msg
      | lookup_msg
      | list_msg
      | cancel_msg
      | fee_msg
      | add_spec_msg
      | cannot_confirm_msg
    ]
    * The kinds of messages that are given upon executing registration
      requests that will affect the student's record, financial record,
      and so on. *


with_effect_request (data flow) =
    [ add_course_req
      | confirm_req
      | change_req
      | drop_req
      | lookup_req
      | list_req
      | cancel_req
      | fee_req
      | add_spec_req
    ]
    * The kinds of requests that can be made to the system that will
      not affect the student's record, financial record, and so on. *


withhold_flag (control flow, del) =
    [ "TRUE"
      | "FALSE"
    ]
    * This flag is set to "TRUE" if the student has a financial hold or
      a disciplinary block tagged to her record.  Otherwise the
      flag is set to "FALSE". *
```

Appendix C

Threads-based Specification

A threads-based natural language specification
of the SIS-Telereg System
Marie Hélène Wong Cheng In
February 21, 1994

C.1 Introduction

C.1.1 Purpose

This document is written to specify the requirements of the **Telereg**[1] system which has been in existence since 1988. It is expected that the use of this document will be valuable in the maintenance of the system. This document will only be concerned with the functionality of a subset of the **Student Information System** that is related to the **Telereg** system.

C.1.2 Background

Before the **Student Information System (SIS)**, front-loaded with the **Telereg** system was installed, registration at the University of British Columbia, UBC, was done manually. In the pre-1988 days, registration took place during the week after Labor Day at the War Memorial Gymnasium, UBC. Each student would pick up an "*authorization to register*" form and collect course cards or signatures from department representatives for each course she wished to enrol in. The student would then have her schedule approved by her school or major department. The "*authorization to register*" forms would then be returned to the Registrar at the gymnasium. Each night of registration week, the course cards would then be fed into a card reader to be loaded into the **Student Record System (SRS)**. The **SRS** was an archaic system that was unable to meet the growing needs of the Registrar's office and the University.

The **Student Information System** is the new system at UBC that is designed to establish a single database for student information. It allows decentralised update and inquiry access with appropriate access controls, and provides convenient interfaces to other centralised administrative systems. The **Student Information System** also provides data to support comprehensive reporting and analysis as well as the flexibility to implement new policies and support changing needs of the University, faculties and departments. **Telereg** is UBC's telephone registration system which was introduced in 1988. Telereg is the student's interface to the **SIS**. Telereg allows students to register for their courses from any touch-tone telephone. The students use the buttons on the telephone keypad to enter requests and the computer's voice guides them through their registration.

[1] *TELEREG*$_{TM}$ is a registered trademark of the University of British Columbia (UBC).

C.1.3 Sources of Information and Acknowledgements

Sources of Information

- Doug Loewen, Senior Technical Analyst, Administrative Technical Support, University Computing Services, UBC.
- Maureen Elliott, Administrative Supervisor, Records and Registration, Office of the Registrar, UBC.
- Beth Smith, Programmer Analyst, Production Systems, University Computing Services, UBC.
- IBM 3270 Information Display System Component Description GA27-2749-10.
- IBM Dialog manual.
- The University of British Columbia 93/94 Calendar
- The University of British Columbia 93/94 Registration Guide
- The VOCOM-40 brochure from Perception Technology, USA.
- Elaine Wright, B.C.Tel Representative for UBC.

Acknowledgements

I wish to thank Maureen Elliott, Beth Smith, and Doug Loewen for their invaluable time and their willingness to meet with me on numerous occasions to explain the intricacies of the SIS-Telereg system. They have been unselfish with their time and have provided me with useful information. Elaine Wright is also to be thanked for her help in the technical aspects of telephone communications. My gratitude goes to Carol Whitehead and Jean Forsythe who have helped me in their own ways to understand the SIS-Telereg system better.

C.1.4 Organization

The established context for this project consists of physical, peripheral, hardware, and software components described in Section C.2. Section C.3 specifies the functional requirements of the system while Section C.4 presents the non-functional requirements of the system. Section C.5 specifies the kinds of exceptional events that may occur and how the system responds to them. Section C.6 describes how the system is expected to evolve in the future. Section C.7 specifies the method used to validate the system. The data dictionary can be found in the appendix.

C.2 Context

The established context for the project includes all of the physical, peripheral, hardware, and software components of the Telereg system. Section C.2.1 of this document describes the physical aspects of the established context. The peripheral and hardware contexts are specified in Section C.2.2, and Section C.2.3 provides a detailed description of the immediate context of the project, that is the software interface.

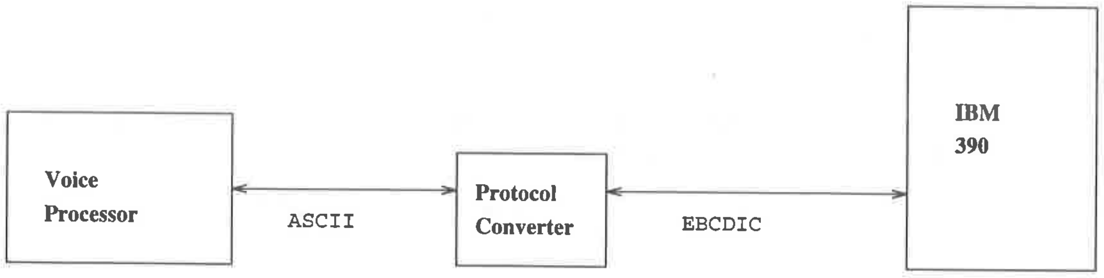


Figure C.73: Linking the peripheral components to the hardware component

C.2.1 Physical Context

The physical components of the established context consists of a touch-tone telephone and a business line with an overline service in which calls are automatically completed to the next available line of a group of 48 lines when the dialed number is busy.

Touch-tone Telephone

The touch-tone telephone keypad and receiver provide the input and output devices respectively to the student. Each button on touch-tone phones makes a different tone (as specified by some telecommunications standard) when pressed. This allow combinations of sequences of touch tones. Verbal messages are used to request for information or to provide feedback on execution of request.

Business Line

The business line has a maximum of 48 overlines. Telereg offers one business telephone number to call and the telephone company distributes that one phone line to the first free overline. When there are too many calls from an area, they are blocked by the telephone company at its discretion even though not all the lines may be used.

C.2.2 Peripheral and Hardware Contexts

The peripheral components of the established context consists of a VOCOM-40 voice processor and a protocol converter while the hardware component consists solely of an IBM 390 **Multiple Virtual System** (MVS) mainframe.

Figure C.73 shows how the various components of the peripheral and hardware contexts are linked together. The voice processor interfaces directly with the IBM MVS machine through the protocol converter. The functions of each of these components will be explained in the following three sections.

VOCOM-40 Voice Processor

The main two components of the VOCOM-40 system are the BT-III Voice Processor and the Digital Speech Recorder (DSR). An additional component, the Application Processor (A/P) will be installed and Section C.6.2 on page 294 describes how its addition to the existing configuration can enhance the system.

The BT-III Voice Processor is a DEC PDP-11, model 73 machine. The BT-III Voice Processor is responsible for phone communications, touch-tone recognition, and speech. It operates 48 phone lines. The model has a capacity of 48 lines within one box and can be linked with another BT-III box for a total expansion to 96 lines. The BT-III is currently configured with a capacity of 30 hours of speech and is expandable to 390 hours. It is capable of sending or receiving voice messages. Incoming messages can be recorded like an answering machine or like voice mail. Each outgoing message is digitised and stored on disk, and is referenced by a unique "message number". Individualised messages that are transmitted to the caller are composed at run time by assembling together shorter messages that are referenced by their unique "message number". The software on the IBM specifies the message numbers of the shorter messages that will compose the complete message and the order in which they will appear. The Voice Processor composes the message and outputs it verbally.

The BT-III supervises calls and is capable of transferring or initiating calls. The Voice Processor has some built-in commands such that it can pick up the phone, hang up the line or interrupt its speech if touch tones are received. However, it will execute such commands only when and as directed by the software running on the IBM machine. The IBM indicates that a command is to be executed through a character and possibly a line number. The BT-III has built-in logic that associates the character with an action. When a user calls the business line, the BT-III detects the ring and informs the IBM. The software on the IBM 390 machine will determine whether to answer the phone and will pass on the letter command to execute to the BT-III and the line number. As such, the Voice Processor knows nothing of the Telereg application. It can give out messages and accept input from the telephone. It also converts the touch tones to ASCII data stream.

The Digital Speech Recorder is used to record and edit speech, and download speech to the BT-III.

Details of the protocol between the Voice Processor and the protocol converter are given in the brochure distributed by the manufacturer of the Voice Processor on the VOCOM-40. The connection between the Voice Processor to the protocol converter is a Null Serial Cable.

IBM 390 Mainframe

Both the SIS and Telereg system software reside on an IBM 390 MVS mainframe. There are 48 communication channels on the IBM reserved for the Telereg application. Multiplexing between the Voice Processor and the IBM is based on the line number that is generated by the Voice Processor. Data from one channel on the IBM is made into a packet which includes the line number provided by the channel. The connection between the IBM to the protocol converter is Ethernet.

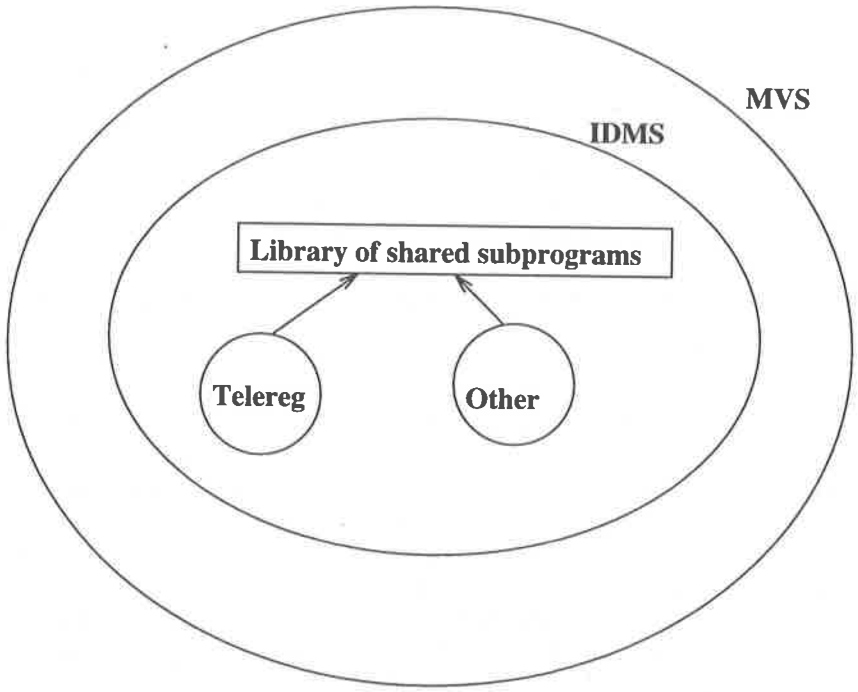


Figure C.74: Software context

Protocol Converter

The protocol converter is actually a program that runs on a PC. It converts ASCII data stream from the Voice Processor to an EBCDIC datastream in 3270 format, as specified in document GA27-2749-10, IBM 3270 Information Display System Component Description, to the IBM machine, and vice versa. The protocol converter, in essence, allows the Voice Processor to communicate with the hardware component. Since there are 48 possible lines in the Voice Processor, the protocol converter checks the line number associated with the data in order to send the converted data to the appropriate channel at the IBM or telephone connection depending on the direction in which the data came from. Communication over the protocol converter is full duplex and the rate of conversion is limited by the speed of the line. It converts data either way and does one conversion at a time. Data from the Voice Processor is queued at the Voice Processor and data from the IBM is queued at the IBM if there is contention for data conversion from either side. Data from the protocol converter is meant to look like terminal I/O to the IBM channel.

C.2.3 Software Context

The IBM 390 mainframe runs the **Multiple Virtual System** (MVS) operating system which runs the product **Integrated Database Management System** (IDMS), an operating system for database management. Within IDMS, various SIS applications including Telereg are run (see figure C.74). These applications share the same database and some of the same subprograms. The only distinguishing feature between these applications is their front-end programs. The front-end programs handle the input and

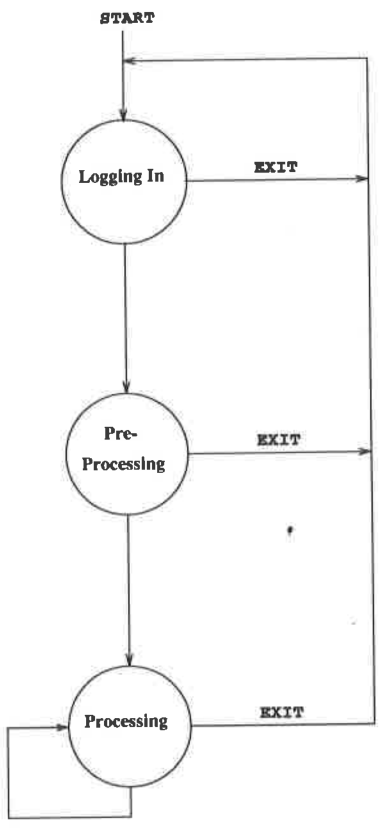


Figure C.75: Modes of operation

output data and call the appropriate subroutines to execute the request. The front-end programs do not interact with each other. Telereg gives students the means to access a limited set of functions of the SIS system. When IDMS is started, all applications are started as well.

C.3 Functional Requirements

The system can be viewed as a finite state machine whereby, at any instant, it is in a particular mode of operation and within a mode of operation be in a particular state.

Figure C.75 shows the transitions from one mode of operation to the next. The "Logging In" mode is always the starting mode of operation. From the "Logging In" mode, the telephone registration system can either proceed to the "Preprocessing" mode or exit. From the "Preprocessing" mode, the system can move on to the "Processing" mode or exit. Finally, from the "Processing" mode of operation, the system can either exit to the starting mode of operation or go back to the same mode of operation.

Figure C.76 shows in detail the transition from states to states within each mode of operation and from one mode of operation to another. The "Logging In" mode is always the starting mode of operation and, more specifically, the "Waiting_for_identification-_input" state is always the starting state. The system stays in the starting state if

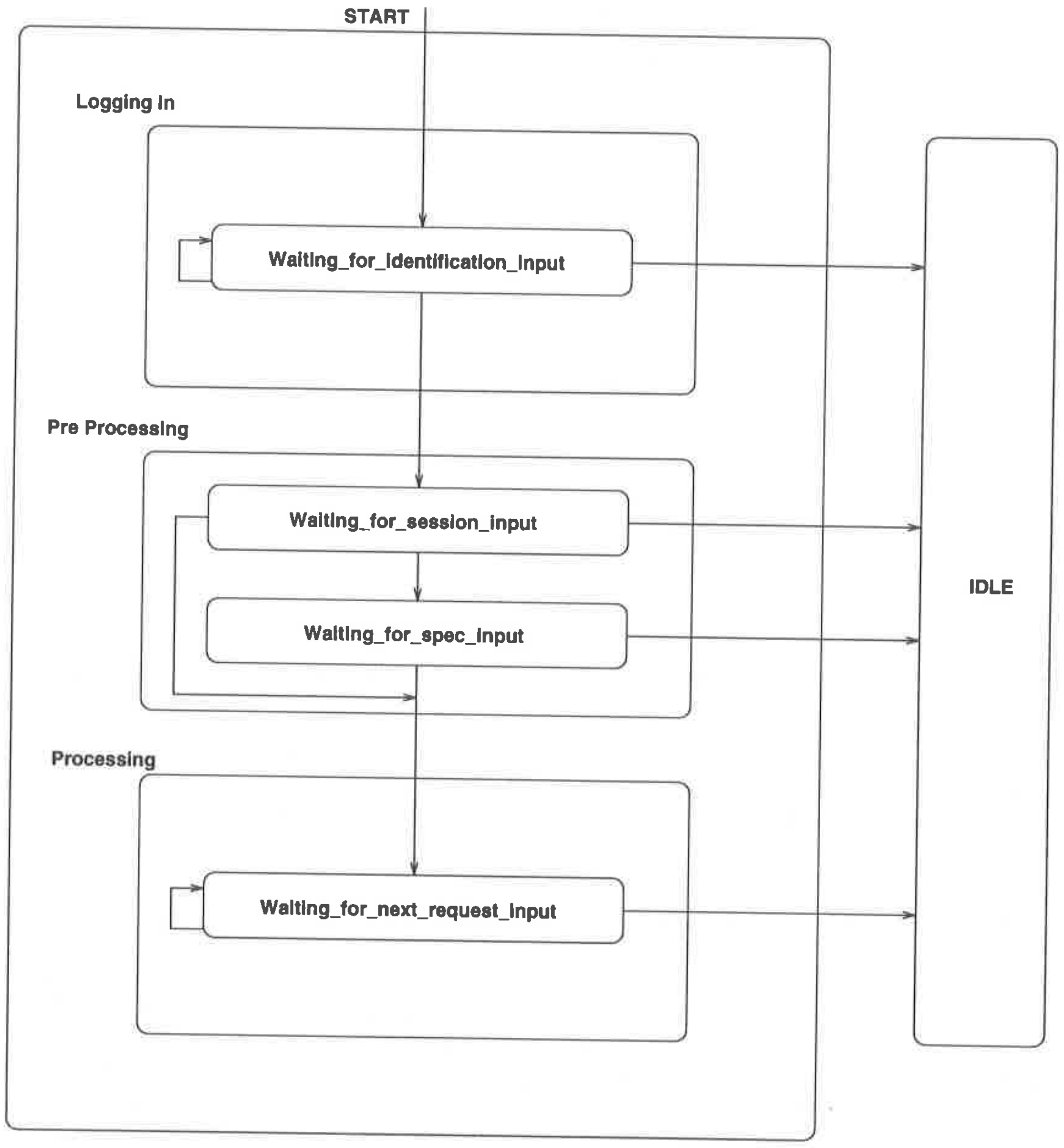

START
Logging In
Waiting_for_identification_input
Pre Processing
Waiting_for_session_input
Waiting_for_spec_input
Processing
Waiting_for_next_request_input
IDLE

Figure C.76: States within modes of operation

the student identification is invalid and it is not the student's third attempt at entering her identification. The system moves to the "IDLE" state if it is the student's third unsuccessful attempt at entering the student identification or the student has financial holds or disciplinary blocks on her record. If the student identification is validated and there are no holds or blocks on the student's record, the system moves to the "Waiting_for_session_input" state in the "Preprocessing" mode of operation.

In "Preprocessing" mode, the system moves from "Waiting_for_session_input" state to the "IDLE" state if the student is not to access Telereg for the session that she requested or her eligibility to register for the session has not been approved. If the student is able to access Telereg and is eligible to register and her specialization is available in her student record, the system proceeds to the "Waiting_for_next_request_input" state in the "Processing" mode of operation. However, if the student is able to access Telereg and is eligible to register but a specialization is needed for the student and it is not available in her student record, then the system proceeds to the "Waiting_for_spec_input" state. The system moves to "Waiting_for_next_request_input" state from "Waiting_for_spec_input" state when the student's specialization is entered and is available on file.

In "Processing" mode, the system stays in the "Waiting_for_next_request_input" state if the input request is invalid and it is not the student's third attempt at entering a request. If it is the student's third unsuccessful attempt at entering a request, there is transition to the "IDLE" state. If the input request is valid, Telereg executes the request. The system proceeds to the "IDLE" state if the exit request was the input request, otherwise it stays in the "Waiting_for_next_request_input" state for any other input requests.

If the student hangs up the phone, transition is always made to the "IDLE" state irrespective of the state and mode of operation of the system. In any state, the student can request for the last message to be repeated.

C.3.1 Validate Identification Thread

OVERVIEW:

This thread describes the processing performed upon receiving a student identification. Telereg will validate the input and if it is accepted, the first four letters of the student's last name are spelled out. It is also checked whether the student has financial holds or disciplinary blocks on her record or an outstanding fee payment.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input student identification] of the form:
a. <student_id>

RESPONSE:

Telereg shall return a [student identification acceptance] as follows:
a. <last_name_msg>

Telereg shall return an [invalid identification rejection] as follows:
a. <invalid_id_msg>

Telereg shall return an [identification request] as follows:
a. <request_id_msg>

Telereg shall return a [connection rejection] as follows:
a. <bye_msg>

Telereg shall return a [hold or block rejection] as follows:
a. <hold_or_block_msg>

Telereg shall return an [outstanding fee rejection] as follows:
a. <fee_kickout_msg>

REQUIREMENTS:

If the system is in the "Waiting_for_identification_input" state, Telereg will check whether the [input student identification] can be found in the <student records>.

Telereg shall return a [connection rejection] provided all of the following conditions are true:

a. 'The input cannot be found in the <student records>'.

b. 'It is the student's third attempt at entering her student identification'.

Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

Telereg shall return an [invalid identification rejection] and an [identification request] provided all of the following conditions are true:

a. 'The input cannot be found in the <student records>'.

b. 'It is not the student's third attempt yet at entering her student identification'.

Telereg will stay in "Waiting_for_identification_input" state.

Telereg shall return a [hold or block rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'There are <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

The telephone connection shall be terminated. No further processing will be done. Telereg shall commit the system state to "IDLE" state.

Telereg shall return an [outstanding fee rejection] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'The student has an <outstanding fee balance>' as specified in the \Check Outstanding Fee Assessment Condition\.

Telereg will then only allow the student to ask for transactions such as current fee assessment and inquiry on section openings.

Telereg shall return a [student identification acceptance] if all of the following conditions are true:

a. 'The input can be found in the <student records>'.

b. 'There are no <financial holds> or <disciplinary blocks> on the student's record' as specified in the \Withhold Student Registration Condition\.

Telereg shall commit to the "Waiting_for_session_input"state.

C.3.2 Withhold Student Registration Condition

OVERVIEW:

This condition describes when the student will be withheld from registering.

REQUIREMENTS:

If there are any <disciplinary blocks> or <financial holds> that have been placed on the student's record, she will be withheld from registering.

C.3.3 Check Outstanding Fee Assessment Condition

OVERVIEW:

This condition describes when the student has an <outstanding fee balance> that will only allow her to ask for a limited set of registration transactions.

REQUIREMENTS:

The student has an <outstanding fee balance> if she failed to make payment on a past due date. Telereg will then only allow the student to ask for transactions such as current fee assessment and inquiry on section openings.

C.3.4 Compute Current Fee Assessment Thread

OVERVIEW:

This thread describes the processing performed upon receiving a fee request. Telereg will compute the current tuition and fees balance owed by the student. If a session is not provided, Telereg will use the previous session as the default session for which to calculate the fee balance. Even though a student may not be allowed to register for a session, Telereg will allow her to query her fee balance for the previous session.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input fee balance request] of the form:
    a. <fee_request>

RESPONSE:

Telereg shall return a [fee balance feedback] as follows:
    a. <fee_msg>

REQUIREMENTS:

If Telereg is in either "Waiting_for_session_input" state or "Waiting_for_next_request_input" state, Telereg will compute the current <tuition and fees balance> owed by the student based on courses she has enrolled in for the <session> included in [input fee balance request] and the payments already made to the university. Telereg shall return a [fee balance feedback] based on these computations and the dates on which instalment payments are due.

Telereg will stay in its initial state.

C.3.5 Preprocess Registration Thread

OVERVIEW:

This thread describes the processing performed upon receiving the session for which the student wishes to register. This thread determines whether a student will be allowed to access the registration process on an attempt and if so, whether she is eligible to register for a session after her student identification has been validated in the list of <student records> at the university. If the student is able to access Telereg and is eligible to register, Telereg will return a feedback of the session for which registration is sought. Telereg will return the specialization that the student is registered for if the specialization is available on file, otherwise it will request for a specialization from the student if one is needed.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input session] of the form:
  a. <session>

RESPONSE:

Telereg shall return a [session feedback] as follows:
  a. <session_msg>

Telereg shall return a [register attempt rejection] as follows:
  a. <cannot_register_msg>

Telereg shall return a [specialization request] as follows:
  a. <ask_spec_msg>

Telereg shall return a [specialization feedback] as follows:
  a. <say_spec_msg>

REQUIREMENTS:

Upon the receipt of an [input session] in the "Waiting_for_session_input" state, Telereg will check the student's accessibility to the system and eligibility to register for the <session> included in [input session].

Telereg shall return a [register attempt rejection] if at least one of the following conditions is true:

a. 'The student is not to access the registration process for the <session> included in [input session]' as specified in \Check Access Dates Condition\.

b. 'The student is not eligible to register for courses in the <session> included in [input session]' as specified in \Check Eligibility To Register Condition\.

Telereg shall terminate the telephone connection. No further processing will take place. Telereg shall commit to the "IDLE" state.

Telereg shall return a [session feedback] and a [specialization feedback] if all of the following conditions are true:

a. 'The student can access the registration process for the <session> included in [input session]' as specified in \Check Access Dates Condition\.

b. 'The student is eligible to register for courses in the <session> included in [input session]' as specified in \Check Eligibility To Register Condition\.

c. 'The student <specialization> is available on file'.

Telereg shall commit to "Waiting_for_next_request_input" state.

Telereg shall return a [session feedback] and a [specialization request] if all of the following conditions are true:

a. 'The student can access the registration process for the <session> included in [input session]' as specified in \Check Access Dates Condition\.

b. 'The student is eligible to register for courses in the <session> included in [input session]' as specified in \Check Eligibility To Register Condition\.

c. 'The student's <specialization> is not available on file and the student's <specialization> is needed'.

Telereg shall commit to "Waiting_for_spec_input" state.

C.3.6 Check Access Dates Condition

OVERVIEW:

This condition specifies when the student will be allowed access to the registration process for the session requested. Each student is assigned an access date based on her year, degree program, category, previous year session average or admission average. That access date is the first day that the student can access a particular session via Telereg.

REQUIREMENTS:

The student is not to have access to the registration process if she is attempting to register at a date prior to the first accessible date set for her by the Registrar's office.

C.3.7 Check Eligibility To Register Condition

OVERVIEW:

This condition specifies when a student is eligible to register. A student has an eligibility to register if she is a student newly admitted to the university or she is a continuing student whose academic performance in the last session meets university requirements for continuing an academic career at the university or she is a continuing student who has been approved to a program by a department.

REQUIREMENTS:

A student is not eligible to register if she does not meet any of the registration eligibility criteria.

C.3.8 Invalid Request Rejection Condition

OVERVIEW:

This condition specifies the conditions under which Telereg will invalidate a registration request and terminate the telephone connection.

REQUIREMENTS:

A registration request will be deemed invalid if any one of the following conditions is true:

a. 'The student's record is being updated by another user'.

b. All of the following conditions are true:

  1. 'The registration request is invalid'.

  2. 'The <maximum number of timeouts> has been exceeded'.

The telephone connection will be terminated.

C.3.9 Try Again Request Condition

OVERVIEW:

This condition specifies the conditions under which Telereg will invalidate a registration request but the student will be given another chance at entering the registration request.

REQUIREMENTS:

A registration request will be deemed invalid but the student will be given another chance at entering the registration request if all the following conditions are true:

a. 'The [input register request] is invalid'.

b. 'The <maximum number of timeouts> has not been exceeded'.

C.3.10 Check Validity Of Requests Capability

OVERVIEW:

This capability describes the processing performed when the student enters a registration request. Telereg will check the validity of the request and ask for the request again if it is not valid and the maximum number of timeouts has not been exceeded. Telereg will exit if the maximum number of timeouts is exceeded. Telereg will inform the student of her number of free transactions left if there are less than ten of them.

OUTPUTS:

Telereg shall return a [invalid request rejection] as follows:
a. <invalid_request_msg>

Telereg shall return a [try again request] as follows:
a. <try_again_msg>

Telereg shall return a [free transactions left feedback] as follows:
a. <free_transactions_left_msg>

Telereg shall return a [fee charge feedback] as follows:
a. <fee_charged_msg>

Telereg shall return a [maximum charge fee rejection] as follows:
a. <maximum_charge_fee_msg>

REQUIREMENTS:

Telereg shall return an [invalid request rejection] if 'the request is invalid and the connection is to be terminated' as specified in \Invalid Request Rejection Condition\. Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

Telereg shall return a [try again request] if 'the request is invalid but the student is to be given another chance at entering the registration request' as specified in \Try Again Request Condition\. Telereg will stay in 'Waiting_for_next_request_input" state.

If the registration request is valid,

a. Telereg shall return a [free transactions left feedback] if 'the student has more than zero but fewer than ten <free transactions> left to use up'.

b. If 'the <maximum charge fee> is exceeded', Telereg shall return a [maximum charge fee rejection]. Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

c. If 'the <maximum charge fee> is not exceeded' and 'the student is to be charged a <transaction fee>' as specified in \Charge Fee For Transaction If Necessary Condition\, Telereg shall charge the student a <transaction fee>. Telereg shall return a [fee charge feedback] to the student. Telereg shall commit the student's number of <free transactions> to one less.

C.3.11 Charge Fee For Transaction If Necessary Condition

OVERVIEW:

This thread describes the condition when Telereg will charge a transaction fee to the student's account.

REQUIREMENTS:

Telereg will charge a <transaction fee> if the student has used up all of her <free transactions> and is requesting for any of adding a section, dropping a section, changing a section, or looking up an inquiry on a section.

C.3.12 Execute Exit Request Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to exit the system. Telereg will compute the student's fee balance and give payment due dates.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input exit request] of the form:
a. <exit_req>

RESPONSE:

Telereg shall return an [exit feedback] as follows:
a. <exit_msg>

REQUIREMENTS:

If Telereg is in the "Waiting_for_next_request_input" state, Telereg shall validate the [input exit request] as specified in \Check Validity Of Requests Capability\.

Telereg will calculate the student <fee assessment> based on previous and current transactions made to the <student schedule> and the amounts and dates by which payments are due. Telereg shall return an [exit feedback] based on those amounts and dates. Telereg shall terminate the telephone connection. No further processing will be done. Telereg shall commit to the "IDLE" state.

C.3.13 Repeat Last Message Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to repeat the last message. Telereg will repeat the last message to the student.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input repeat request] of the form:
   a. <repeat_req>

RESPONSE:

Telereg shall return a [repeat feedback] as follows:
   a. <repeat_msg>

REQUIREMENTS:

In any state that Telereg may be in, Telereg shall return a [repeat feedback] according to the <last message> it gave.

C.3.14 Add A Section Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to add a section to the student's schedule.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input add section] of the form:
   a. <add_section_req>

RESPONSE:

Telereg shall return an [add section feedback] as follows:
   a. <add_section_msg>

Telereg shall return an [add section rejection] as follows:
   a. <cannot_add_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input add section] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>', Telereg shall return an [add section feedback] provided all of the following conditions are true:

a. 'The student is not registered in a <Standard TimeTable>'.

b. 'There are <openings> in the <section> included in [input add section]'.

c. 'The student will not have exceeded the <maximum number of credits> that she is allowed to register in with the addition of the <section> included in [input add section]'.

d. 'The student is not already registered in the <section> included in [input add section]'.

If all of the above conditions are true:

a. Telereg shall commit the student to the <class list> for the <section> included in [input add section].

b. Telereg shall commit the number of <openings> for the <section> included in [input add section] to one less.

c. Telereg shall commit the <section> included in [input add section] to the <student schedule>.

If any one of the above conditions is not true, Telereg shall return an [add section rejection].

C.3.15 Confirm A Section Switch Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to confirm a section switch in the student's schedule.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input confirm switch] of the form:
a. <confirm_req>

RESPONSE:

Telereg shall return a [confirm switch feedback] as follows:
a. <confirm_msg>

Telereg shall return a [confirm switch rejection] as follows:
a. <cannot_confirm_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input confirm switch] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>', Telereg shall return a [confirm switch feedback] if all of the following conditions are true:

a. 'The student is not registered in a <Standard TimeTable>'.

b. 'The <section> included in [input confirm switch] can be found in the <student schedule>'.

If at least one of the above conditions is not true, Telereg shall return a [confirm switch rejection].

C.3.16 Change Section Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to change a section in the student's schedule.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input change section] of the form:
a. <change_req>

RESPONSE:

Telereg shall return a [change section feedback] as follows:
a. <change_msg>

Telereg shall return a [change section rejection] as follows:
a. <cannot_change_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input change section] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>', Telereg will perform a <section> switch in the <student schedule> provided all of the following conditions are true:

a. 'The student is not registered in a <Standard TimeTable>'.

b. 'There are <openings> in the new <section> included in [input change section]'.

c. 'The old <section> to be removed as derived from the <section> included in [input change section] can be found in the <student schedule>'.

If any one of the above conditions is not true, Telereg shall return a [change section rejection].

If all of the above conditions are true,

a. Telereg shall commit the new <section> included in [input change section] in the <student schedule>.

b. Telereg shall commit to the removal of the old <section> as derived from the <section> included in [input change section] from the <student schedule>.

c. Telereg shall commit the number of <openings> for the old <section> as derived from the <section> included in [input change section] to one more.

d. Telereg shall commit the number of <openings> for the new <section> included in [input change section] to one less.

e. Telereg shall commit the student to the <class list> of the new <section> included in [input change section].

f. Telereg shall commit to the removal of the student from the <class list> of the old <section> as derived from the <section> included in [input change section].

g. Telereg shall return a [change section feedback].

C.3.17 Drop Section Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to drop a section from the student's schedule.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input drop section] of the form:
a. <drop_req>

RESPONSE:

Telereg shall return a [drop section feedback] as follows:
a. <drop_msg>

Telereg shall return a [drop section rejection] as follows:
a. <cannot_drop_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input drop section] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>', Telereg shall return a [drop section feedback] if all of the following conditions are true:

a. 'The student is not registered in a <Standard TimeTable>'.

b. 'The <section> included in [input drop section] can be found in the <student schedule>'.

If any one of the above conditions is not true, Telereg shall return a [drop section rejection].

If all of the above conditions are true,

a. Telereg shall commit to the removal of the <section> included in [input drop section] from the <student schedule>.

b. Telereg shall commit to the removal of the student from the <class list> associated with the <section> included in [input drop section].

c. Telereg shall commit the <number of openings> for the <section> included in [input drop section] to one more.

C.3.18 Look Up Inquiry On Sections Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to look up the number of openings in a section.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input lookup section] of the form:
a. <lookup_req>

RESPONSE:

Telereg shall return a [lookup section feedback] as follows:
a. <lookup_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input lookup section] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid', Telereg will look up the <number of openings> for the <section> included in [input lookup section]. Telereg shall return a [lookup section feedback] based on the results of that lookup.

C.3.19 List All Courses Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to list all the sections in the student's schedule.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input list all] of the form:
a. <list_req>

RESPONSE:

Telereg shall return a [list all feedback] as follows:
a. <list_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input list all] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>', Telereg shall return a [list all feedback] based on the list of sections in the <student schedule>.

C.3.20 Cancel Entire Registration Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to cancel the student's entire registration.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input cancel all] of the form:

a. <cancel_req>

RESPONSE:

Telereg shall return a [cancel all feedback] as follows:
a. <cancel_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state, Telereg shall validate the [input cancel all] as specified in \Check Validity Of Requests Capability\.

If 'the request is valid' and 'the student does not have an <outstanding fee balance>' and 'the student is not registered under the <Standard TimeTable>',

a. Telereg shall commit to the removal of each of the <section>s that the student has registered for from the <student schedule>.

b. Telereg shall commit to the removal of the student from the <class list> associated with each <section>.

c. Telereg shall commit the <number of openings> for each <section> to one more.

d. Telereg shall return a [cancel all feedback].

C.3.21 Add Specialization Thread

OVERVIEW:

This thread describes the processing performed upon receiving a request to add a specialization to the student's record.

STIMULUS:

Telereg shall satisfy the requirements of this thread upon the receipt of an [input add specialization] of the form:
a. <add_spec_req>

RESPONSE:

Telereg shall return an [add specialization feedback] as follows:
a. <add_spec_msg>

Telereg shall return an [invalid specialization rejection] as follows:
a. <invalid_spec_msg>

REQUIREMENTS:

In the "Waiting_for_next_request_input" state or "Waiting_for_spec_input" state, Telereg will validate the <specialization> included in [input add specialization].

Telereg shall return a [invalid specialization rejection] if 'the <specialization> included in [invalid specialization rejection] is not valid'. Telereg will stay in its initial state.

If 'the <specialization> included in [invalid specialization rejection] is valid', Telereg shall add the [input add specialization] to the student's record. Telereg shall return an [add specialization feedback]. Telereg shall commit to "Waiting_for_next_request_input" state if its initial state is "Waiting_for_spec_input".

C.3.22 Handle Hang Up Capability

OVERVIEW:

This capability describes the processing performed upon detection of a hang up.

REQUIREMENTS:

In any state that Telereg may be in, when Telereg detects a hang up, Telereg shall complete the transaction it was executing if it is in the midst of executing a transaction. Telereg shall commit to the "IDLE" state.

C.4 Non-Functional Requirements

C.4.1 Product Requirements

Compiled code for the program must fit within the memory size of the program store. The required software is implemented in the ADS/O COBOL language. In addition, the software conforms to the coding standard specified by the customer.

C.4.2 Response Time

Since response time is a continuing concern, it has been monitored since Telereg was first introduced and various performance tuning projects over the years have been initiated to improve program efficiency. Response time is directly related to the amount of processing done by each command and the load on the system. To improve the response time, every program in the Telereg system was tuned and a better mainframe was installed. The system was designed such that the processing load is minimized and no one command does too much processing. For instance, data is not validated more than once and validation occurs only when the data is needed.

On average, a call usually lasts four minutes. If the response time between the student completing a command entry and the response to that command is greater than

60 seconds, then steps are taken to minimize the load. These remedial steps include any of the following: shutting down some of the 48 phone lines, minimizing other activities on the mainframe such as giving development programmers a lower priority for CPU cycles, and restricting the use of certain heavy user commands such as the "Compute Current Fee Assessment" command during the peak hours of the day. Part of the data for the whole **Student Information System** is kept in tables known as "**YES/NO tables**". By looking up the "YES/NO" value of an entry, a subroutine will either enable or disable the completion of its execution. The "Compute Current Fee Assessment" function checks the table entry associated with it before performing the command. By simply setting the table entry associated with the "Compute Current Fee Assessment" function to "no", its use will be restricted.

C.4.3 Reliability

The software needs to recover gracefully if exceptional events occur. To decrease the probability of failure, three programming environments are used. They are each dedicated for a specific purpose: development, testing, and production. All new software is tested in the testing environment which includes a full-sized database and two Telereg phone lines. The new software needs to be approved by the Registrar's Office before it goes into the production environment.

C.4.4 Security

All administrative systems, including the SIS and Telereg, are situated on a separate mainframe which is physically isolated from all other UBC computer systems. All connections to this mainframe are in physically secure locations. Identifications and passwords are required to connect to the system and there are several levels of security to control which screens are available to each user. If a screen is not accessible by a user, it will not appear on that user's menu. A further level of security controls which actions are available to a user on a given screen.

The primary security check that the system performs for Telereg callers is on birth dates. In essence, the birth date is a pin number. The student may request that the Registrar's office change her pin number. The software will check that the given pin number matches the pin number on file for that student.

All changes to data are logged to both SIS specific database save areas and IDMS specific journal files which are later copied to tapes.

C.4.5 Concurrency

Concurrency is achieved by running 48 instantiations of the Telereg front-end programs for each channel. This implies that, at any one time, the Telereg system can support up to 48 simultaneous users which include both students and Registrar personnel. This maximum number of users can be increased by getting a greater number of overlines from the local telephone company and running more instantiations of the Telereg front-end program.

C.5 Exceptional Events

Hardware overheating, hardware component failure, application software bugs, and telecommunication failures have been some of the reasons causing the Telereg software to crash over the years. Since crashing software is inevitable, the Telereg software has been designed such that, in the best case, when the software detects a problem it tells the student "There has been a Telereg system error. Please call again later. Goodbye." In the worst case, the software crashes and the telephone connection is terminated.

To ensure that the software recovers gracefully, the Voice Processor enables an "application down" routine and answers the lines with an advisory message and then terminates the telephone connection if it is still running. At the time of a crash, all completed commands from a given phone session will have been processed and saved, but commands in progress may not have taken effect. It is up to students to check their last command when they phone back.

To minimize downtime in the event of problems, various spare hardware components are on site. Operators who are available at all times to monitor the Telereg system have a list of people to contact in case problems occur.

Two common exceptional events occur when a registration session is not terminated by the exit sequence or the touch tones are fuzzy. In the former case, the voice response unit detects such events as hangups and inactivity timeouts and notifies the mainframe of these events. The application software is coded to handle a hangup. In the latter case, anything other than proper touch tones would not be understood and would be equivalent to inactivity and result in a timeout condition.

C.6 System Evolution

C.6.1 Software Evolution

It can be expected that the software can be further modified to allow verification of time conflicts in a student's registration schedule. Also, at this time, the software does not check whether pre-requisites and co-requisites are met before allowing a student to register in a course. It is deemed that computer time to check the pre-requisites and co-requisites of each and every course would greatly slow down a telephone registration session by increasing the response time.

C.6.2 Peripheral Evolution

The Application Processor is one of the three components of the VOCOM-40 voice unit that has not been installed yet. The Application Processor is a 486 PC that runs interactive UNIX. Interactive UNIX allows multiple users, multi-tasking, multiple host connections, and multiple applications. The Application Processor is responsible for application control and host communications (if connected to a host). An application generator, ScriptPower, can be installed on the Application Processor and be used to generate voice processing applications via easy-to-use pull-down menus.

After installation, it will be connected between the Voice Processor and the protocol converter. It is expected that, in the long run, applications with telephone access other than Telereg will be preprocessed on the Application Processor before they are properly dispatched to the IBM mainframe to the appropriate application program. In the case of the Telereg application, the addition of the Application Processor will alleviate time resource requirements on the mainframe because most validations on data content and format can be performed on it.

C.7 Validation Criteria

The customer will provide an acceptance test in the form of a series of executions of the program under varying conditions and test cases to test its functionality and its limits.

Data Dictionary

```
<add_section_msg> =
    * This is the feedback that is given by the system upon executing
      a request to add a section to a student's schedule for
      a particular session to confirm that the request has been
      executed. *

<add_section_req> =
    * This request consists of a command to add a section and the
      section to be added. *

<add_spec_msg> =
    * This is the feedback that is given by the system upon executing
      a request to add a specialization or degree program to the
      student's record to confirm that the request has been executed. *

<add_spec_req> =
    * This request consists of a command to add a specialization
      or degree program and the specialization or degree program
      to be added. *

<ask_spec_msg> =
    * This is a message to prompt for a specialization or degree
      program if either one is found missing in the student's record. *

<bye_msg> =
    * This is a message that is given by the system if the student
      fails to enter a valid student identification within three
      attempts. The system will not allow the student to proceed. *

<cancel_msg> =
    * This is the feedback that is given by the system upon executing
      a request to cancel the student's entire registration for a
      particular session to confirm that the request has been executed. *

<cancel_req> =
    * This request consists of a command to cancel the student's entire
      registration for a particular session. *

<cannot_add_msg> =
    * This is a message that is given when the system fails to add
      the requested section to the student's schedule because of many
      possible reasons: the class limit for the section has been
      reached, the requested section is an invalid one, the student
      has already registered for the section, and so on. *

<cannot_change_msg> =
    * This is a message that is given when the system is not able
```

```
        to change a section as requested by the student.  This might be
        because that section is full, the specified section is an
        invalid one, and so on. *

<cannot_confirm_msg> =
    * This is a message that is given when the system is not able to
      confirm a section switch as requested by the student.
      This might be because the student has not previously registered
      in that section or the student has a Standard TimeTable which
      implies that she cannot change her schedule, and so on. *

<cannot_drop_msg> =
    * This is a message that is given when the system is not able to
      drop a section as requested by the student. This might be
      because the student has not previously registered in that
      section or the student has a Standard TimeTable which implies
      that she cannot change her schedule, and so on. *

<cannot_register_msg> =
    * This is a message that is given if the student is attempting to
      register before the time specified by her access date
      or if the student is not eligible to register for courses in
      the session specified because of a low grade average. *

<change_msg> =
    * This is the feedback that is given by the system upon executing
      a request to change a section in the student's schedule
      to confirm that the request has been executed.  *

<change_req> =
    * This request consists of a command to change a section and
      the section to change to. *

<class list> =
    * Each section has a class list associated with it.  A class list
      is the list of students who have registered in a section. *

<confirm_msg> =
    * This is the feedback that is given by the system upon executing
      a request to confirm that a section switch has been made. *

<confirm_req> =
    * This request consists of a command to confirm that a section
      switch has been made and the section to be checked. *

<disciplinary blocks> =
    * These are restrictions that are placed on a student's record
      to prevent her from registering.  These restrictions arise because
      of academic reasons such as failure to keep in good academic
      standing and so on. *

<drop_msg> =
    * This is the feedback that is given by the system upon executing
      a request to drop a section from a student's course schedule to
      confirm that the request has been executed. *

<drop_req> =
    * This request consists of a command to drop a section and the
      section to be dropped. *

<exit_msg> =
    * This is the feedback that is given by the system upon executing
      a request to exit the registration session. *

<exit_req> =
    * This request consists of a command to exit the registration
      process. *

<fee assessment> =
    * This is the student's tuition and fees balance in terms of the
      amounts owed and their specific due dates. *

<fee_charged_msg> =
    * This message is given out to indicate to the student that a
      fee was charged for the request that she had made.  A fee is
      charged for a request if the student has already used up the
      maximum number of free transaction requests that she is allowed
      for each session. *

<fee_kickout_msg> =
    * This message is given by the system if it finds that the
      student who is attempting to register has an outstanding
      financial account.  The student will only be able to ask for
      a limited set of transaction requests. *

<fee_msg> =
    * This is the feedback given by the system upon executing
      a request to inquire about a student's current fee assessment.
      The dollar amounts owed and the dates on which they
      are due are given. *

<fee_request> =
    * This request consists of a command to inquire about a particular
      student's current fee assessment. *

<financial holds> =
    * These are restrictions that are placed on a student's record
      to prevent her from registering.  These restrictions arise because
      of financial reasons such as an outstanding financial balance
      or any other reasons that would require for the student to
      clear with the financial departments first before she is allowed
      to register. *

<free transactions> =
    * Use of Telereg is free for the first 60 transactions.  A
      student should be able to complete her registration within
      this arbitrary number of transactions that is determined by the
      Registrar's Office.  Once that number is exceeded, the student
      is charged a fee for each of her transactions.  Beyond a maximum
      charge fee, the student is restricted from registering. *

<free_transactions_left_msg> =
    * If a student has used her free transactions requests so that
      she is only eligible for fewer than ten free transactions
      in a session, then this message will warn her of the number of
      free transactions left. *

<hold_or_block_msg> =
    * This is a message that is given by the system to indicate that
      the student has a financial hold or a disciplinary block on her
      record and the appropriate party she should contact for more
      information. *

<invalid_id_msg> =
    * This is the message that is given if the student_id provided
      is found to be invalid. *

<invalid_request_msg> =
    * This is a message that is given if the request that the student
      provided is found to be a faulty one or an invalid one or if
      the student's record is being concurrently updated by another user. *

<invalid_spec_msg> =
    * This is the message that is given if the specialization
      provided by the student is an invalid one. *

<last message> =
    * This is the last message that the system has given to the
      student. *

<last_name_msg> =
    * This personalized message for each student who is trying to
      register gives the first four letters of the student's
      last name after the student gives her student_id. *

<list_msg> =
    * This is the feedback that is given by the system upon executing
```

```
      a request to list all the courses that a student has
      registered for in a particular session. *

<list_req> =
    * This request consists of a command to list all the sections
      that the student has registered for in a particular session. *

<lookup_msg> =
    * This is the feedback that is given by the system upon executing
      a request to look up availability in a particular section. *

<lookup_req> =
    * This request consists of a command to inquire about availability
      in a particular section and the section to inquire about. *

<maximum charge fee> =
    * See entry for <free_transactions>. *

<maximum_charge_fee_msg> =
    * The message that is given by the system to the student to inform
      her that she will not be allowed to register because she has
      exceeded the maximum charge fee. *

<maximum number of credits> =
    * Each section is associated with a certain number of credits.
      Depending on her degree program, a student will not be allowed
      to register for more than the maximum number of credits specified
      by the department that she belongs to. *

<maximum number of timeouts> =
    * The maximum number of times that the system will allow a student
      to enter a registration request because either the student's
      request is invalid or the student has failed to enter a
      complete request before a timeout occurred. *

<number of openings> =
    * This is the number of students who can still register for a
      section before the class size limit is reached. *

<openings> =
    * See entry for <number of openings>. *

<outstanding fee balance> =
    * The student has an outstanding fee balance if she has failed
      to make payment on a past due date. *

<request_id_msg> =
    * This is a message to request the user for a student_id. *
```

```
<register_request> =
    <add_course_req> |
    <add_spec_req> |
    <cancel_re> |
    <change_req> |
    <confirm_req> |
    <drop_req> |
    <exit_req> |
    <fee_request> |
    <list_req> |
    <lookup_req>
    * The kinds of registration requests that can be made to the
      system during a registration session. *

<repeat_msg> =
    * This is the feedback that is given by the system upon executing
      a request to repeat the previous message. *

<repeat_req> =
    * This request consists of a command to repeat the previous
      message. *

<say_spec_msg> =
    * This message is given as a feedback to the student to indicate
      the specialization or degree program for which she is
      admitted to. *

<section> =
    * A section corresponds to a particular course offered at a
      particular time and taught by a particular instructor.  *

<session> =
    * The academic year for which the student wishes to register for. *

<session_msg> =
    * The message that is given by the system as a feedback to
      indicate the session for which the student wishes to register in. *

<specialization>  =
    * This is the degree program or specialization for which the student
      is eligible. *

<student_id> =
    student_number
    + birth_date
    * The identification that a student provides in order to register. *

<student records> =
    * The complete list of files that the system has on the
```

```
      students at the university. *

<student schedule> =
    * This is the list of sections that the student has registered in
      a session. *

<Standard TimeTable> =
    * A number of sections that have been grouped together to form a
      conflict free registration package required for a specific
      program.  Students are automatically registered in each
      of the courses in a Standard TimeTable if they belong to a
      specific program. *

<transaction fee> =
    * See entry for <free_transactions>. *

<try_again_msg> =
    * The message given by the system to allow the student
      another chance to input the registration request. *

<tuition and fees balance> =
    * The student's financial balance of what she owes the university
      for tuition and fees. *
```

Appendix D

Z Specification

A Z specification of
the SIS-Telereg System
Marie Hélène Wong Cheng In
March 2, 1993

Note

1. The Z specification may be improved by determining the preconditions of the partial operations specified.
2. Some schemas are specified in two parts so that all the information is included in the appendix. They must be combined when the specification is type-checked.

Global Types

This section describes all the different kinds of data objects in the system and some basic types.

$[SId]$

$[Section]$

$[Course]$

$[Session]$

$[Balance]$

$[Date]$

$[Specialization]$

$RegistrationType ::= STT\_Schedule \mid Regular$

$Schedule == \mathbb{P}\, Section$

$Openings == \mathbb{Z}$

$Classlist == \mathbb{P}\, SId$

$Credits == \mathbb{Z}$

$Counter == \mathbb{Z}$

$Free\_Transactions == \mathbb{Z}$

$LockModes ::= LOCKED \mid UNLOCKED$

$Register\_Requests ::= ADD \mid DROP \mid LOOKUP \mid CHANGE$
$\quad \mid EXIT \mid CONFIRM \mid LIST \mid CANCEL \mid ADDSPEC$

$$
\begin{array}{l}
\mathit{Stimuli\_Or\_Responses} ::= \\
\quad \mathit{add\_section\_msg} \mid \\
\quad \mathit{add\_section\_req} \langle\!\langle \mathit{Section} \rangle\!\rangle \mid \\
\quad \mathit{add\_spec\_msg} \langle\!\langle \mathit{Specialization} \rangle\!\rangle \mid \\
\quad \mathit{add\_spec\_req} \langle\!\langle \mathit{Specialization} \rangle\!\rangle \mid \\
\quad \mathit{ask\_spec\_msg} \mid \\
\quad \mathit{bye\_msg} \mid \\
\quad \mathit{cancel\_msg} \mid \\
\quad \mathit{cancel\_rejection} \mid \\
\quad \mathit{cancel\_req} \mid \\
\quad \mathit{cannot\_add\_msg} \mid \\
\quad \mathit{cannot\_change\_msg} \mid \\
\quad \mathit{cannot\_confirm\_msg} \mid \\
\quad \mathit{cannot\_drop\_msg} \mid \\
\quad \mathit{cannot\_register\_msg} \mid \\
\quad \mathit{change\_msg} \mid \\
\quad \mathit{change\_req} \langle\!\langle \mathit{Section} \rangle\!\rangle \mid \\
\quad \mathit{confirm\_msg} \mid \\
\quad \mathit{confirm\_req} \langle\!\langle \mathit{Section} \rangle\!\rangle \mid \\
\quad \mathit{drop\_msg} \mid \\
\quad \mathit{drop\_req} \langle\!\langle \mathit{Section} \rangle\!\rangle \mid \\
\quad \mathit{exit\_msg} \langle\!\langle \mathit{Balance} \rangle\!\rangle \mid \\
\quad \mathit{exit\_req} \langle\!\langle \mathit{Session} \rangle\!\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{Stimuli\_Or\_Responses} ::= \\
\quad \mathit{fee\_charged\_msg} \mid \\
\quad \mathit{fee\_kickout\_msg} \mid \\
\quad \mathit{fee\_msg}\langle\!\langle \mathit{Balance} \times \mathit{Session} \rangle\!\rangle \mid \\
\quad \mathit{fee\_request}\langle\!\langle \mathit{Session} \rangle\!\rangle \mid \\
\quad \mathit{free\_transactions\_left\_msg}\langle\!\langle \mathit{Counter} \rangle\!\rangle \mid \\
\quad \mathit{hangup} \mid \\
\quad \mathit{hold\_or\_block\_msg} \mid \\
\quad \mathit{invalid\_id\_msg} \mid \\
\quad \mathit{invalid\_request\_msg}\langle\!\langle \mathit{Register\_Requests} \rangle\!\rangle \mid \\
\quad \mathit{invalid\_spec\_msg} \mid \\
\quad \mathit{last\_name\_msg} \mid \\
\quad \mathit{list\_msg}\langle\!\langle \mathit{Course} \rangle\!\rangle \mid \\
\quad \mathit{list\_req} \mid \\
\quad \mathit{lookup\_msg}\langle\!\langle \mathit{Openings} \rangle\!\rangle \mid \\
\quad \mathit{lookup\_req}\langle\!\langle \mathit{Section} \rangle\!\rangle \mid \\
\quad \mathit{maximum\_charge\_fee\_msg} \mid \\
\quad \mathit{request\_id\_msg} \mid \\
\quad \mathit{repeat\_msg}\langle\!\langle \mathit{Stimuli\_Or\_Responses} \rangle\!\rangle \mid \\
\quad \mathit{repeat\_req} \mid \\
\quad \mathit{say\_spec\_msg}\langle\!\langle \mathit{Specialization} \rangle\!\rangle \mid \\
\quad \mathit{session}\langle\!\langle \mathit{Session} \rangle\!\rangle \mid \\
\quad \mathit{session\_msg}\langle\!\langle \mathit{Session} \rangle\!\rangle \mid \\
\quad \mathit{student\_id}\langle\!\langle \mathit{SId} \rangle\!\rangle \mid \\
\quad \mathit{try\_again\_msg}
\end{array}
$$

$System\_State ::=$

$\quad Waiting\_for\_identification\_input \mid$

$\quad Waiting\_for\_session\_input \mid$

$\quad Waiting\_for\_spec\_input \mid$

$\quad Waiting\_for\_next\_request\_input \mid$

$\quad IDLE$

$\_ \geq \_ : Balance \leftrightarrow Balance$

$\_ < \_ : Date \leftrightarrow Date$

Global Constants

This section describes the static attributes of the system.

$Telereg\_State$

$Students : \mathbb{P}\, SId$

$RegisteredIn : SId \rightarrow Schedule$

$Availability : Section \rightarrow Openings$

$Class : Section \rightarrow Classlist$

$ScheduleType : SId \rightarrow RegistrationType$

$Blocks : \mathbb{P}\, SId$

$Holds : \mathbb{P}\, SId$

$Outstandings : \mathbb{P}\, SId$

$ValidSession : \mathbb{P}\, Session$

$Eligibles : SId \rightarrow Session$

$Accessed : SId \rightarrow Date$

$Account : SId \rightarrow Balance$

$LockStatus : SId \rightarrow LockModes$

$ChargeFeeStatus : SId \rightarrow Balance$

$state : System\_State$

$count : Free\_Transactions$

$StudentSpec : SId \rightarrow Specialization$

$ValidSpec : \mathbb{P}\, Specialization$

$MAX\_TIMEOUT\_COUNT : \mathbb{Z}$

Telereg_State

$MAX\_ID\_COUNT : \mathbb{Z}$

$max\_credits : \mathbb{Z}$

$MAX\_CHARGE\_FEE : Balance$

$Commands : \mathbb{P}\, Register\_Requests$

$CostlyCommands : \mathbb{P}\, Register\_Requests$

$NumberCredits : Section \rightarrow \mathbb{Z}$

$TotalCredits : SId \rightarrow \mathbb{Z}$

$CourseSchedule : Section \rightarrow Course$

$(\mathrm{dom}\; RegisteredIn) \subseteq Students$

$Blocks \subseteq Students$

$Holds \subseteq Students$

$Outstandings \subseteq \mathrm{dom}\; Account$

$Outstandings \subseteq Students$

$CostlyCommands \subseteq Commands$

Axiomatic Definitions

The student is registered in a Standard Time Table.

$$\begin{array}{|l}
\textit{InSTTSchedule} \\
\hline
\textit{Telereg\_State} \\
id : SId \\
\hline
\textit{ScheduleType}\ id = \textit{STT\_Schedule} \\
\hline
\end{array}$$

The student is registered in a regular time table.

$$\begin{array}{|l}
\textit{InRegularSchedule} \\
\hline
\textit{Telereg\_State} \\
id : SId \\
\hline
\textit{ScheduleType}\ id = \textit{Regular} \\
\hline
\end{array}$$

Telereg shall commit the student to the class list for the section.

$$\begin{array}{|l}
\textit{AddToClassList} \\
\hline
\Delta \textit{Telereg\_State} \\
id : SId \\
section : Section \\
newclasslist : Classlist \\
\hline
newclasslist = Class\ section \cup \{id\} \\
Class' = Class \oplus \{section \mapsto newclasslist\} \\
\hline
\end{array}$$

Telereg shall commit to the removal of the student from the class list of the section.

RemoveFromClassList

$$\begin{array}{l}
\Delta \mathit{Telereg\_State} \\
\mathit{id} : \mathit{SId} \\
\mathit{section} : \mathit{Section} \\
\mathit{newclasslist} : \mathit{Classlist} \\
\hline
\mathit{newclasslist} = \mathit{Class}\ \mathit{section} \setminus \{\mathit{id}\} \\
\mathit{Class}' = \mathit{Class} \oplus \{\mathit{section} \mapsto \mathit{newclasslist}\}
\end{array}$$

Telereg shall commit the number of openings for the section to one less.

DecrementOpenings

$$\begin{array}{l}
\Delta \mathit{Telereg\_State} \\
\mathit{section} : \mathit{Section} \\
\mathit{newopenings} : \mathit{Openings} \\
\hline
\mathit{newopenings} = \mathit{Availability}\ \mathit{section} - 1 \\
\mathit{Availability}' = \mathit{Availability} \oplus \{\mathit{section} \mapsto \mathit{newopenings}\}
\end{array}$$

Telereg shall commit the number of openings for the section to one more.

IncrementOpenings

$$\begin{array}{l}
\Delta \mathit{Telereg\_State} \\
\mathit{section} : \mathit{Section} \\
\mathit{newopenings} : \mathit{Openings} \\
\hline
\mathit{newopenings} = \mathit{Availability}\ \mathit{section} + 1 \\
\mathit{Availability}' = \mathit{Availability} \oplus \{\mathit{section} \mapsto \mathit{newopenings}\}
\end{array}$$

Telereg shall commit the section to the student's schedule.

AddToSchedule

$\Delta\, Telereg\_State$

$id : SId$

$section : Section$

$newschedule : Schedule$

$newschedule = RegisteredIn\ id \cup \{section\}$

$RegisteredIn' = RegisteredIn \oplus \{id \mapsto newschedule\}$

Telereg shall commit to the removal of the section from the student's schedule.

RemoveFromSchedule

$\Delta\, Telereg\_State$

$id : SId$

$section : Section$

$newschedule : Schedule$

$newschedule = RegisteredIn\ id \setminus \{section\}$

$RegisteredIn' = RegisteredIn \oplus \{id \mapsto newschedule\}$

There are openings in the section.

ThereAreOpenings

$Telereg\_State$

$section : Section$

$Availability\ section > 0$

The student will not have exceeded the maximum number of credits that she is allowed to register in with the addition of the section.

$$\begin{array}{|l}
\textit{MaxCreditsNotExceeded} \\
\hline
\textit{Telereg\_State} \\
\textit{id} : \textit{SId} \\
\textit{section} : \textit{Section} \\
\hline
\textit{TotalCredits}\ \textit{id} + \textit{NumberCredits}\ \textit{section} \leq \textit{max\_credits} \\
\hline
\end{array}$$

The student is enrolled in the section.

$$\begin{array}{|l}
\textit{StudentEnrolledInSection} \\
\hline
\textit{Telereg\_State} \\
\textit{id} : \textit{SId} \\
\textit{section} : \textit{Section} \\
\hline
\textit{section} \in (\textit{RegisteredIn}\ \textit{id}) \\
\hline
\end{array}$$

The student is withdrawn from the section, which entails that the student is removed from the classlist of the section, the section is removed from the student's schedule, and the number of openings in the section is incremented by one.

$$\begin{array}{|l}
\textit{WithdrawStudentFromSection} \\
\hline
\Delta \textit{Telereg\_State} \\
\textit{id} : \textit{SId} \\
\textit{section} : \textit{Section} \\
\textit{newschedule} : \textit{Schedule} \\
\textit{newclasslist} : \textit{Classlist} \\
\textit{newopenings} : \textit{Openings} \\
\hline
\textit{RemoveFromSchedule} \\
\textit{RemoveFromClassList} \\
\textit{IncrementOpenings} \\
\hline
\end{array}$$

The student is registered in the section, which entails that the student is added to the classlist of the section, the section is added to the student's schedule, and the number of openings in the section is decremented by one.

```
┌─ RegisterStudentInSection ──────────────────────────
│ ΔTelereg_State
│ id : SId
│ section : Section
│ newschedule : Schedule
│ newclasslist : Classlist
│ newopenings : Openings
├──────────
│ AddToSchedule
│ AddToClassList
│ DecrementOpenings
└─────────────────────────────────────────────────────
```

Withhold Student Registration Condition

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition describes when the student will be withheld from registering.

Conditions and Modes

If there are any disciplinary blocks or financial holds that have been placed on the student's record, she shall be withheld from registering.

$$
\begin{array}{|l}
\hline
\textit{WSRC\_holds\_or\_blocks\_condition} \\
\hline
\textit{Telereg\_State} \\
id : SId \\
\hline
id \in Holds \lor id \in Blocks \\
\hline
\end{array}
$$

Actions

None.

Requirements

None.

Check Outstanding Fee Assessment Condition

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition describes when the student has an outstanding fee balance that will only allow her to ask for a limited set of registration transactions.

Conditions and Modes

The student has an outstanding fee balance if she failed to make payment on a past due date.

$$
\begin{array}{|l}
\textit{COFAC\_outstanding\_fee\_condition} \\
\hline
\textit{Telereg\_State} \\
id : SId \\
\hline
id \in Outstandings \\
\hline
\end{array}
$$

Actions

None.

Requirements

None.

Validate Identification Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 11 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. WSRC_holds_or_blocks_condition as specified in \Withhold Student Registration Condition\
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a student identification. Telereg will validate the input and if it is accepted, the first four letters of the student's last name are spelled out. It is also checked whether the student has financial holds or disciplinary blocks on her record, as well as whether the student has an outstanding fee payment.

Stimuli

```
┌─ VIT_input_student_identification ─────────────────
│ Telereg_State
│ id : SId
│ input? : Stimuli_Or_Responses
├──────────
│ input? = student_id(id)
└────────────────────────────────────────────────────
```

Responses

$VIT\_student\_identification\_acceptance$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = last\_name\_msg$

$VIT\_invalid\_identification\_rejection$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = invalid\_id\_msg$

$VIT\_identification\_request$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = request\_id\_msg$

$VIT\_connection\_rejection$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = bye\_msg$

$VIT\_hold\_or\_block\_rejection$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = hold\_or\_block\_msg$

$VIT\_outstanding\_fee\_rejection$

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = fee\_kickout\_msg$

Conditions and Modes

The student identification can be found in the student records.

$VIT\_id\_found\_condition$

$Telereg\_State$

$id : SId$

$id \in Students$

It is the student's third attempt at inputting her student identification.

$VIT\_max\_attempt\_exceeded\_condition$

$Telereg\_State$

$id\_count : \mathbb{Z}$

$id\_count \geq MAX\_ID\_COUNT$

There are financial holds or disciplinary blocks on the student's record.

$$VIT\_withhold\_condition \mathrel{\widehat=} WSRC\_holds\_or\_blocks\_condition$$

The student has an outstanding fee balance.

$$VIT\_outstanding\_fee\_condition \mathrel{\widehat=} COFAC\_outstanding\_fee\_condition$$

The system is in "Waiting_for_identification_input" state.

$$VIT\_in\_identification\_input\_state \mathrel{\widehat=}$$
$$[Telereg\_State \mid state = Waiting\_for\_identification\_input]$$

Actions

Telereg shall commit to "IDLE" state.

$$VIT\_move\_to\_idle\_state \mathrel{\widehat=}$$
$$[\Delta Telereg\_State \mid state' = IDLE]$$

Telereg shall commit to "Waiting_for_session_input" state.

$$VIT\_move\_to\_session\_input\_state \mathrel{\widehat=}$$
$$[\Delta Telereg\_State \mid state' = Waiting\_for\_session\_input]$$

Requirements

In the "Waiting_for_identification_input" state, Telereg shall terminate the telephone connection if the student identification cannot be found in the student records and it is the student's third attempt at inputting her student identification. Telereg shall commit to the "IDLE" state.

$$\begin{aligned}
&VIT\_Requirement1 \mathrel{\widehat=}\\
&\quad VIT\_input\_student\_identification \land\\
&\quad VIT\_in\_identification\_input\_state \land\\
&\quad \neg\ VIT\_id\_found\_condition \land\\
&\quad VIT\_max\_attempt\_exceeded\_condition \Rightarrow\\
&\quad VIT\_connection\_rejection \land\\
&\quad VIT\_move\_to\_idle\_state
\end{aligned}$$

In the "Waiting_for_identification_input" state, Telereg shall inform the student that the student identification is invalid and ask for another identification if the student identification cannot be found in the student records and it is not the student's third attempt at inputting her student identification.

$$
\begin{aligned}
&\mathit{VIT\_Requirement2} \mathrel{\widehat=} \\
&\quad \mathit{VIT\_input\_student\_identification} \land \\
&\quad \mathit{VIT\_in\_identification\_input\_state} \land \\
&\quad \neg\ \mathit{VIT\_id\_found\_condition} \land \\
&\quad \neg\ \mathit{VIT\_max\_attempt\_exceeded\_condition} \Rightarrow \\
&\quad \mathit{VIT\_invalid\_identification\_rejection} \land \\
&\quad \mathit{VIT\_identification\_request}
\end{aligned}
$$

In the "Waiting_for_identification_input" state, Telereg shall terminate the telephone connection after informing the student that she has a financial hold or disciplinary block on her record if her student identification can be found in the student records but there is also a hold or block on the student's record. Telereg shall commit to the "IDLE" state.

$$
\begin{aligned}
&\mathit{VIT\_Requirement3} \mathrel{\widehat=} \\
&\quad \mathit{VIT\_input\_student\_identification} \land \\
&\quad \mathit{VIT\_in\_identification\_input\_state} \land \\
&\quad \mathit{VIT\_id\_found\_condition} \land \\
&\quad \mathit{VIT\_withhold\_condition} \Rightarrow \\
&\quad \mathit{VIT\_hold\_or\_block\_rejection} \land \\
&\quad \mathit{VIT\_move\_to\_idle\_state}
\end{aligned}
$$

In the "Waiting_for_identification_input" state, after Telereg finds the student identification in the student records, it shall inform the student that she has an outstanding fee balance if she has one.

$$
\begin{aligned}
&\mathit{VIT\_Requirement4} \mathrel{\widehat=} \\
&\quad \mathit{VIT\_input\_student\_identification} \land \\
&\quad \mathit{VIT\_in\_identification\_input\_state} \land \\
&\quad \mathit{VIT\_id\_found\_condition} \land \\
&\quad \mathit{VIT\_outstanding\_fee\_condition} \Rightarrow \\
&\quad \mathit{VIT\_outstanding\_fee\_rejection}
\end{aligned}
$$

In the "Waiting_for_identification_input" state, Telereg shall spell out the student's last name after the student identification can be found in the student records and the student is not to be withheld because of financial holds or disciplinary blocks. Telereg shall commit to the "Waiting_for_session_input"state.

$$
\begin{array}{l}
VIT\_Requirement5 \mathrel{\widehat=} \\
\quad VIT\_input\_student\_identification \land \\
\quad VIT\_in\_identification\_input\_state \land \\
\quad VIT\_id\_found\_condition \land \\
\quad \neg\ VIT\_withhold\_condition \Rightarrow \\
\quad VIT\_student\_identification\_acceptance \land \\
\quad VIT\_move\_to\_session\_input\_state
\end{array}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{array}{l}
Validate\_Identification\_Thread \mathrel{\widehat=} \\
\quad VIT\_Requirement1 \land \\
\quad VIT\_Requirement2 \land \\
\quad VIT\_Requirement3 \land \\
\quad VIT\_Requirement4 \land \\
\quad VIT\_Requirement5
\end{array}
$$

Compute Current Fee Assessment Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 11 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

None.

Overview

This thread describes the processing performed upon receiving a fee request. Telereg will compute the current tuition and fees balance owed by the student. If a session is not provided, Telereg will use the previous session as the default session for which to calculate the fee balance. Even though a student may not be allowed to register for a session, Telereg will allow her to query her fee balance for the previous session.

Stimuli

```
┌─ CCFAT_input_fee_balance_request ──────────────
│ Telereg_State
│ ss : Session
│ input? : Stimuli_Or_Responses
├──────────
│ input? = fee_request(ss)
└────────────────────────────────────────────────
```

Responses

```
┌─ CCFAT_fee_balance_feedback ──────────────────────
│ ΔTelereg_State
│ id : SId
│ ss : Session
│ output! : Stimuli_Or_Responses
├──────────
│ output! = fee_msg((Account id), ss)
└───────────────────────────────────────────────────
```

Conditions and Modes

Telereg is in "Waiting_for_session_input" state.

$$CCFAT\_in\_session\_input\_state \mathrel{\widehat=} [Telereg\_State \mid state = Waiting\_for\_session\_input]$$

Telereg is in "Waiting_for_next_request_input" state.

$$CCFAT\_in\_next\_request\_input\_state \mathrel{\widehat=} [Telereg\_State \mid state = Waiting\_for\_next\_request\_input]$$

Actions

None.

Requirements

In the "Waiting_for_session_input" state or the "Waiting_for_next_request_input" state, Telereg shall give to the student the amounts owed by her and the due dates for each amount.

$$\begin{aligned}
&Compute\_Current\_Fee\_Assessment\_Thread \mathrel{\widehat=}\\
&\quad CCFAT\_input\_fee\_balance\_request \land\\
&\quad CCFAT\_in\_next\_request\_input\_state \lor\\
&\quad CCFAT\_in\_session\_input\_state \Rightarrow\\
&\quad CCFAT\_fee\_balance\_feedback
\end{aligned}$$

Check Access Dates Condition

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition specifies when the student will be allowed access to the registration process for the session requested. Each student is assigned an access date based on her year, degree program, category, previous year session average or admission average. That access date is the first day that the student can access a particular session via Telereg.

Conditions and Modes

The student is not to have access to the registration process if she is attempting to register at a date prior to the first accessible date set for her by the Registrar's office.

$$
\begin{array}{|l}
\textit{CADC\_no\_access\_condition} \\
\hline
\textit{Telereg\_State} \\
ss : \textit{Session} \\
\textit{currentDate} : \textit{Date} \\
id : \textit{SId} \\
\hline
\textit{currentDate} < \textit{Accessed}\ id \\
\hline
\end{array}
$$

Actions

None.

Requirements

None.

Check Eligibility To Register Condition

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition specifies when a student is eligible to register. A student has an eligibility to register if she is a student newly admitted to the university or she is a continuing student whose academic performance in the last session meets university requirements for continuing an academic career at the university or she is a continuing student who has been approved to a program by a department.

Conditions and Modes

A student is not eligible to register if she does not meet any of the registration eligibility criteria.

$$
\begin{array}{|l}
\hline
\textit{CERC\_no\_eligibility\_condition} \\
\hline
\textit{Telereg\_State} \\
ss : \textit{Session} \\
id : \textit{SId} \\
\hline
\{id \mapsto ss\} \subseteq \textit{Eligibles} \\
\hline
\end{array}
$$

Actions

None.

Requirements

None.

Preprocess Registration Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 11 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. CADC_no_access_condition as specified in \Check Access Dates Condition\
2. CERC_no_eligibility_condition as specifies in \Check Eligibility To Register Condition\

Overview

This thread describes the processing performed upon receiving the session for which the student wishes to register. This thread determines whether a student will be allowed to access the registration process on an attempt and if so, whether she is eligible to register for a session after her student identification has been validated and she is on the list of student records at the university. If the student is able to register, Telereg will return a feedback of the session for which registration is sought. Telereg will return the specialization that the student is registered for if there is one on file, otherwise it will request for one from the student if one is needed.

Stimuli

```
┌─ PRT_input_session ──────────────────────────
│ Telereg_State
│ ss : Session
│ input? : Stimuli_Or_Responses
├──────────
│ input? = session(ss)
└──────────────────────────────────────────────
```

Responses

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{PRT\_session\_feedback}} \\
\Delta \mathit{Telereg\_State} \\
\mathit{ss} : \mathit{Session} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{session\_msg}(\mathit{ss}) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{PRT\_register\_attempt\_rejection}} \\
\Delta \mathit{Telereg\_State} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{cannot\_register\_msg} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{PRT\_specialization\_request}} \\
\Delta \mathit{Telereg\_State} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{ask\_spec\_msg} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{PRT\_specialization\_feedback}} \\
\Delta \mathit{Telereg\_State} \\
\mathit{id} : \mathit{SId} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{say\_spec\_msg}(\mathit{StudentSpec}\ \mathit{id}) \\
\hline
\end{array}$$

Conditions and Modes

The session is valid.

$$\begin{array}{|l}
\textit{PRT\_session\_valid\_condition} \\
\hline
\textit{Telereg\_State} \\
ss : \textit{Session} \\
\hline
ss \in \textit{ValidSession} \\
\hline
\end{array}$$

The student is not to access the registration process.

$$\textit{PRT\_no\_access\_condition} \mathrel{\widehat=} \textit{CADC\_no\_access\_condition}$$

The student is not eligible to register for courses.

$$\textit{PRT\_no\_eligibility\_condition} \mathrel{\widehat=} \textit{CERC\_no\_eligibility\_condition}$$

Telereg is in "Waiting_for_session_input" state.

$$\begin{array}{l}
\textit{PRT\_in\_session\_input\_state} \mathrel{\widehat=} \\
\qquad [\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_session\_input}]
\end{array}$$

The student's specialization is not available on file.

$$\begin{array}{|l}
\textit{PRT\_need\_specialization\_condition} \\
\hline
\textit{Telereg\_State} \\
id : \textit{SId} \\
\hline
\neg\ (\forall\, \textit{spec} : \textit{Specialization} \bullet \\
\qquad \{id \mapsto \textit{spec}\} \subseteq \textit{StudentSpec}) \\
\hline
\end{array}$$

Actions

Telereg shall commit to "Waiting_for_spec_input" state.

$$\begin{array}{l}
\textit{PRT\_move\_to\_spec\_input\_state} \mathrel{\widehat=} \\
\qquad [\Delta\, \textit{Telereg\_State} \mid \textit{state}' = \textit{Waiting\_for\_spec\_input}]
\end{array}$$

Telereg shall commit to "IDLE" state.

$$PRT\_move\_to\_idle\_state \mathrel{\widehat=} \\ [\Delta Telereg\_State \mid state' = IDLE]$$

Telereg shall commit to "Waiting_for_next_request_input" state.

$$PRT\_move\_to\_next\_request\_input\_state \mathrel{\widehat=} \\ [\Delta Telereg\_State \mid state' = Waiting\_for\_next\_request\_input]$$

Requirements

In the "Waiting_for_session_input" state, Telereg shall terminate the telephone connection if the session is valid and the student is not to access Telereg and is not eligible to register. Telereg shall inform her of her status. Telereg shall commit to the "IDLE" state.

$$\begin{aligned} &PRT\_Requirement1 \mathrel{\widehat=} \\ &\quad PRT\_input\_session \land \\ &\quad PRT\_in\_session\_input\_state \land \\ &\quad PRT\_session\_valid\_condition \land \\ &\quad PRT\_no\_access\_condition \lor \\ &\quad PRT\_no\_eligibility\_condition \Rightarrow \\ &\quad PRT\_register\_attempt\_rejection \land \\ &\quad PRT\_move\_to\_idle\_state \end{aligned}$$

In the "Waiting_for_session_input" state, Telereg shall return as feedback the session for which registration is sought as well as the student's specialization if the session is valid and the student is able to access Telereg and is eligible to register and her specialization is available on file. Telereg shall commit to "Waiting_for_next_request_input" state.

$PRT\_Requirement2 \mathrel{\widehat{=}}$

$PRT\_input\_session \land$

$PRT\_in\_session\_input\_state \land$

$PRT\_session\_valid\_condition \land$

$\neg\ PRT\_no\_access\_condition \land$

$\neg\ PRT\_no\_eligibility\_condition \land$

$\neg\ PRT\_need\_specialization\_condition \Rightarrow$

$PRT\_session\_feedback \land$

$PRT\_specialization\_feedback \land$

$PRT\_move\_to\_next\_request\_input\_state$

In the "Waiting_for_session_input" state, Telereg shall return as feedback the session for which registration is sought and shall request the student's specialization if the session is valid and the student is able to access Telereg and is eligible to register and her specialization is not available on file. Telereg shall commit to "Waiting_for_spec_input" state.

$PRT\_Requirement3 \mathrel{\widehat{=}}$

$PRT\_input\_session \land$

$PRT\_in\_session\_input\_state \land$

$PRT\_session\_valid\_condition \land$

$\neg\ PRT\_no\_access\_condition \land$

$\neg\ PRT\_no\_eligibility\_condition \land$

$PRT\_need\_specialization\_condition \Rightarrow$

$PRT\_session\_feedback \land$

$PRT\_specialization\_request \land$

$PRT\_move\_to\_spec\_input\_state$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{aligned}
&\mathit{Preprocess\_Registration\_Thread} \mathrel{\widehat{=}} \\
&\quad \mathit{PRT\_Requirement}1 \land \\
&\quad \mathit{PRT\_Requirement}2 \land \\
&\quad \mathit{PRT\_Requirement}3
\end{aligned}
$$

Invalid Request Rejection Condition

History

Created 16 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition specifies the conditions under which Telereg will invalidate a registration request and terminate the telephone connection.

Conditions and Modes

The maximum number of timeouts has been exceeded.

$$\begin{array}{|l}
\textit{IRRC\_maximum\_timeouts\_exceeded} \\
\hline
Telereg\_State \\
request\_count : \mathbb{Z} \\
\hline
request\_count > MAX\_TIMEOUT\_COUNT \\
\hline
\end{array}$$

The registration request is valid.

$$\begin{array}{|l}
\textit{IRRC\_register\_request\_valid\_condition} \\
\hline
Telereg\_State \\
request : Register\_Requests \\
\hline
request \in Commands \\
\hline
\end{array}$$

The student's record is being updated by another user.

$$\begin{array}{|l}
\textit{IRRC\_concurrent\_access\_condition} \\
\hline
Telereg\_State \\
id : SId \\
\hline
LockStatus\ id = LOCKED \\
\hline
\end{array}$$

A registration request is invalidated if either the student's record is being concurrently updated by another user or the maximum number of timeouts has been exceeded and the request is invalid.

$$\begin{aligned}
&IRRC\_request\_rejection\_condition \mathrel{\widehat=} \\
&\quad IRRC\_concurrent\_access\_condition \lor \\
&\quad \neg\ IRRC\_register\_request\_valid\_condition \land \\
&\quad IRRC\_maximum\_timeouts\_exceeded
\end{aligned}$$

Actions

None.

Requirements

None.

Try Again Request Condition

History

Created 16 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition when Telereg will invalidate a registration request but the student will be given another chance at entering the registration request.

Conditions and Modes

The maximum number of timeouts has been exceeded.

$$\begin{array}{|l}
\hline
\textit{TARC\_maximum\_timeouts\_exceeded} \\
\textit{Telereg\_State} \\
\textit{request\_count} : \mathbb{Z} \\
\hline
\textit{request\_count} > \textit{MAX\_TIMEOUT\_COUNT} \\
\hline
\end{array}$$

The registration request is valid.

$$\begin{array}{|l}
\hline
\textit{TARC\_register\_request\_valid\_condition} \\
\textit{Telereg\_State} \\
\textit{request} : \textit{Register\_Requests} \\
\hline
\textit{request} \in \textit{Commands} \\
\hline
\end{array}$$

A student will be given another chance at entering the registration request if the request is invalid and the maximum number of timeouts has not been exceeded.

Conditions and Modes

$$\begin{aligned}
&\mathit{TARC\_try\_again\_condition} \mathrel{\widehat{=}} \\
&\quad \mathit{TARC\_register\_request\_valid\_condition} \land \\
&\quad \neg\ \mathit{TARC\_maximum\_timeouts\_exceeded}
\end{aligned}$$

Actions

None.

Requirements

None.

Charge Fee For Transaction If Necessary Condition

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Overview

This condition describes when Telereg will charge a transaction fee to the student's account.

Conditions and Modes

Telereg shall charge a transaction fee if the student has used up all of her free transactions and is requesting for any of adding a course, dropping a course, changing a section, or looking up an inquiry on a section.

$$\begin{array}{|l}
\textit{CFTNC\_charge\_transaction\_fee\_condition} \\
\hline
\textit{Telereg\_State} \\
\textit{request} : \textit{Register\_Requests} \\
\hline
\textit{count} \leq 0 \\
\textit{request} \in \textit{CostlyCommands} \\
\hline
\end{array}$$

Actions

None.

Requirements

None.

Check Validity Of Requests Capability

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 11 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. CFTNC_charge_transaction_fee_condition as specified in \Charge Fee For Transaction If Necessary Condition\.
2. IRRC_request_rejection_condition as specified in \Invalid Request Rejection Condition\.
3. TARC_try_again_condition as specified in \Try Again Request Condition\.

Overview

This capability describes the processing performed when a student enters a registration request. Telereg will check the format of the request and ask for the request again if the format is wrong and the maximum number of timeouts has not been exceeded. Telereg will inform the student of the number of free transactions left if there are less than ten of them.

Outputs

$CVRC\_invalid\_request\_rejection$

$\Delta Telereg\_State$

$request : Register\_Requests$

$output! : Stimuli\_Or\_Responses$

$output! = invalid\_request\_msg(request)$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CVRC\_try\_again\_request}} \\
\hline
\Delta \mathit{Telereg\_State} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{try\_again\_msg} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CVRC\_free\_transactions\_left\_feedback}} \\
\hline
\Delta \mathit{Telereg\_State} \\
\mathit{transact\_count} : \mathit{Counter} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{free\_transactions\_left\_msg}(\mathit{transact\_count}) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CVRC\_fee\_charge\_feedback}} \\
\hline
\Delta \mathit{Telereg\_State} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{fee\_charged\_msg} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CVRC\_maximum\_charge\_fee\_rejection}} \\
\hline
\Delta \mathit{Telereg\_State} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{maximum\_charge\_fee\_msg} \\
\hline
\end{array}
$$

Conditions and Modes

The registration request is valid.

```
┌─ CVRC_register_request_valid_condition ─────────
│ Telereg_State
│ request : Register_Requests
├──────────
│ request ∈ Commands
└─────────────────────────────────────────────────
```

The student is to be charged a transaction fee.

$$CVRC\_charge\_transaction\_fee\_condition \mathrel{\widehat{=}} CFTNC\_charge\_transaction\_fee\_condition$$

The registration request is invalid and the maximum number of timeouts has been exceeded or the student's record is being updated by another user.

$$CVRC\_request\_rejection\_condition \mathrel{\widehat{=}} IRRC\_request\_rejection\_condition$$

The registration request is invalid and the maximum number of timeouts has not been exceeded.

$$CVRC\_try\_again\_condition \mathrel{\widehat{=}} TARC\_try\_again\_condition$$

The student has ten or less free transactions left to use.

```
┌─ CVRC_less_than_ten_free_transactions_condition ─────────
│ Telereg_State
├──────────
│ count ≤ 10
└─────────────────────────────────────────────────
```

The maximum charge fee has been exceeded.

```
┌─ CVRC_maximum_charge_fee_exceeded ─────────
│ Telereg_State
│ id : SId
├──────────
│ ChargeFeeStatus id ≥ MAX_CHARGE_FEE
└─────────────────────────────────────────────────
```

Actions

Telereg shall commit to "IDLE" state.

$$CVRC\_move\_to\_idle\_state \mathrel{\widehat{=}}$$
$$[\Delta Telereg\_State \mid state' = IDLE]$$

Telereg shall commit the number of free transactions to one less.

$$\begin{array}{|l}
\textit{CVRC\_decrement\_free\_transactions} \\
\hline
\Delta Telereg\_State \\
\hline
count' = count - 1 \\
\hline
\end{array}$$

Requirements

Telereg shall reject the registration request if either the student's record is being updated by another user or the request is invalid and the maximum number of timeouts has been exceeded. Telereg shall terminate the telephone connection. Telereg shall commit to "IDLE" state.

$$CVRC\_Requirement1 \mathrel{\widehat{=}}$$
$$CVRC\_request\_rejection\_condition \Rightarrow$$
$$CVRC\_invalid\_request\_rejection \land$$
$$CVRC\_move\_to\_idle\_state$$

Telereg shall ask for the registration request again if the request is invalid and the maximum number of timeouts has not been exceeded. Telereg will stay in the same state.

$$CVRC\_Requirement2 \mathrel{\widehat{=}}$$
$$CVRC\_try\_again\_condition \Rightarrow$$
$$CVRC\_try\_again\_request$$

Telereg shall inform the student of the number of free transactions left if the registration request is valid and the student has less than ten free transactions left.

$CVRC\_Requirement3 \mathrel{\widehat{=}}$

$CVRC\_register\_request\_valid\_condition \land$

$CVRC\_less\_than\_ten\_free\_transactions\_condition \Rightarrow$

$CVRC\_free\_transactions\_left\_feedback$

Telereg shall inform the student if the registration request is valid but the maximum charge fee on transactions has been exceeded. Telereg shall commit to the "IDLE" state.

$CVRC\_Requirement4 \mathrel{\widehat{=}}$

$CVRC\_register\_request\_valid\_condition \land$

$CVRC\_maximum\_charge\_fee\_exceeded \Rightarrow$

$CVRC\_maximum\_charge\_fee\_rejection \land$

$CVRC\_move\_to\_idle\_state$

Telereg shall inform the student that a transaction charge fee has been added to her financial balance if the registration request is valid and the maximum charge fee on transactions has not been exceeded and the student has used up all her free transactions. Telereg shall commit the number of free transactions to one less.

$CVRC\_Requirement5 \mathrel{\widehat{=}}$

$CVRC\_register\_request\_valid\_condition \land$

$\neg\ CVRC\_maximum\_charge\_fee\_exceeded \land$

$CVRC\_charge\_transaction\_fee\_condition \Rightarrow$

$CVRC\_fee\_charge\_feedback \land$

$CVRC\_decrement\_free\_transactions$

The specification of this capability is the conjunction of all of the above requirements:

$$\begin{aligned}
&\mathit{Check\_Validity\_Of\_Requests\_Capability} \mathrel{\widehat{=}} \\
&\quad \mathit{CVRC\_Requirement}1 \land \\
&\quad \mathit{CVRC\_Requirement}2 \land \\
&\quad \mathit{CVRC\_Requirement}3 \land \\
&\quad \mathit{CVRC\_Requirement}4 \land \\
&\quad \mathit{CVRC\_Requirement}5
\end{aligned}$$

Execute Exit Request Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.

Overview

This thread describes the processing performed upon receiving a request to exit the system. Telereg will compute the student's fee balance and give payment due dates.

Stimuli

```
┌─ EERT_input_exit_request ──────────────────────────
│ Telereg_State
│ ss : Session
│ input? : Stimuli_Or_Responses
├──────────
│ input? = exit_req(ss)
└────────────────────────────────────────────────────
```

Responses

EERT_exit_feedback

$\Delta Telereg\_State$

$id : SId$

$ss : Session$

$output! : Stimuli\_Or\_Responses$

$output! = exit\_msg(Account\ id)$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$EERT\_in\_next\_request\_input\_state \mathrel{\widehat{=}} [Telereg\_State \mid state = Waiting\_for\_next\_request\_input]$$

The registration request is valid.

$$EERT\_valid\_request\_capability \mathrel{\widehat{=}} Check\_Validity\_Of\_Requests\_Capability$$

Actions

Telereg shall commit to "IDLE" state.

$$EERT\_move\_to\_idle\_state \mathrel{\widehat{=}} [\Delta Telereg\_State \mid state' = IDLE]$$

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. Telereg shall return the student's balance. Telereg shall terminate the telephone connection. Telereg shall commit to the "IDLE" state.

$$
\begin{aligned}
&Execute\_Exit\_Request\_Thread \mathrel{\widehat=} \\
&\quad EERT\_input\_exit\_request \land \\
&\quad EERT\_valid\_request\_capability \land \\
&\quad EERT\_in\_next\_request\_input\_state \Rightarrow \\
&\quad EERT\_exit\_feedback \land \\
&\quad EERT\_move\_to\_idle\_state
\end{aligned}
$$

Repeat Last Message Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

None.

Overview

This thread describes the processing performed upon receiving a request to repeat the last message. Telereg will repeat the last message to the student.

Stimuli

$$
\begin{array}{|l}
\hline
\mathit{RLMT\_input\_repeat\_request} \\
\mathit{Telereg\_State} \\
\mathit{input?} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{input?} = \mathit{repeat\_req} \\
\hline
\end{array}
$$

Responses

$$
\begin{array}{|l}
\hline
\mathit{RLMT\_repeat\_feedback} \\
\Delta \mathit{Telereg\_State} \\
\mathit{lastmsg} : \mathit{Stimuli\_Or\_Responses} \\
\mathit{output!} : \mathit{Stimuli\_Or\_Responses} \\
\hline
\mathit{output!} = \mathit{repeat\_msg}(\mathit{lastmsg}) \\
\hline
\end{array}
$$

Conditions and Modes

None.

Actions

None.

Requirements

Telereg shall repeat the last message it gave to the student.

$$\begin{aligned}
&\mathit{Repeat\_Last\_Message\_Thread} \mathrel{\widehat=} \\
&\quad \mathit{RLMT\_input\_repeat\_request} \Rightarrow \\
&\quad \mathit{RLMT\_repeat\_feedback}
\end{aligned}$$

Add A Section Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to add a section to the student's schedule.

Stimuli

$$
\begin{array}{|l}
\textit{ACT\_input\_add\_section} \\
\hline
\textit{Telereg\_State} \\
\textit{section} : \textit{Section} \\
\textit{input?} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{input?} = \textit{add\_section\_req}(\textit{section}) \\
\hline
\end{array}
$$

Responses

$$
\begin{array}{|l}
\textit{ACT\_add\_section\_feedback} \\
\hline
\Delta \textit{Telereg\_State} \\
\textit{output!} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{output!} = \textit{add\_section\_msg} \\
\hline
\end{array}
$$

$$\begin{array}{|l}
\textit{ACT\_add\_section\_rejection} \\
\hline
\Delta \textit{Telereg\_State} \\
\textit{output!} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{output!} = \textit{cannot\_add\_msg} \\
\hline
\end{array}$$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$\textit{ACT\_in\_next\_request\_input\_state} \mathrel{\widehat=} [\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_next\_request\_input}]$$

The registration request is valid.

$$\textit{ACT\_valid\_request\_capability} \mathrel{\widehat=} \textit{Check\_Validity\_Of\_Requests\_Capability}$$

The student has an outstanding fee balance.

$$\textit{ACT\_outstanding\_fee\_condition} \mathrel{\widehat=} \textit{COFAC\_outstanding\_fee\_condition}$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the student does not have an outstanding fee balance and the request is valid and the student is not registered in a Standard TimeTable and there are openings in the course and the student will not have exceeded the maximum number of credits that she is allowed to register in with the addition of the course and the student is not already registered in the section, the student is added to the classlist for that course and the number of openings in the course is decremented by one. The course is also added to the student's registration schedule for the session.

$$
\begin{array}{l}
ACT\_Requirement1 \mathrel{\widehat=} \\
\quad ACT\_input\_add\_section \land \\
\quad ACT\_in\_next\_request\_input\_state \land \\
\quad ACT\_valid\_request\_capability \land \\
\quad \lnot\ ACT\_outstanding\_fee\_condition \land \\
\quad \lnot\ InSTTSchedule \land \\
\quad ThereAreOpenings \land \\
\quad MaxCreditsNotExceeded \land \\
\quad \lnot\ StudentEnrolledInSection \Rightarrow \\
\quad RegisterStudentInSection \land \\
\quad ACT\_add\_section\_feedback
\end{array}
$$

If any of the conditions described above is not true, then the request to add a section will be rejected.

$$
\begin{array}{l}
ACT\_Requirement2 \mathrel{\widehat=} \\
\quad ACT\_input\_add\_section \land \\
\quad \lnot\ ACT\_in\_next\_request\_input\_state \lor \\
\quad \lnot\ ACT\_valid\_request\_capability \lor \\
\quad ACT\_outstanding\_fee\_condition \lor \\
\quad InSTTSchedule \lor \\
\quad \lnot\ ThereAreOpenings \lor \\
\quad \lnot\ MaxCreditsNotExceeded \lor \\
\quad StudentEnrolledInSection \Rightarrow \\
\quad ACT\_add\_section\_rejection
\end{array}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{aligned}
&Add\_A\_Course\_Thread \mathrel{\widehat=} \\
&\qquad ACT\_Requirement1 \land \\
&\qquad ACT\_Requirement2
\end{aligned}
$$

Confirm A Section Switch Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to confirm a section switch in the student's schedule.

Stimuli

CSST_input_confirm_switch

$Telereg\_State$

$section : Section$

$input? : Stimuli\_Or\_Responses$

$input? = confirm\_req(section)$

Responses

CSST_confirm_switch_feedback

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = confirm\_msg$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CSST\_confirm\_switch\_rejection}} \\
\Delta \textit{Telereg\_State} \\
\textit{output!} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{output!} = \textit{cannot\_confirm\_msg} \\
\hline
\end{array}
$$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$\textit{CSST\_in\_next\_request\_input\_state} \mathrel{\widehat=}$$
$$[\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_next\_request\_input}]$$

The registration request is valid.

$$\textit{CSST\_valid\_request\_capability} \mathrel{\widehat=} \textit{Check\_Validity\_Of\_Requests\_Capability}$$

The student has an outstanding fee balance.

$$\textit{CSST\_outstanding\_fee\_condition} \mathrel{\widehat=} \textit{COFAC\_outstanding\_fee\_condition}$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the request is valid and the student does not have an outstanding fee balance, Telereg shall confirm the section switch if the student is not registered in a Standard TimeTable and the section can be found in the student's schedule.

$$
\begin{array}{l}
\mathit{CSST\_Requirement1} \mathrel{\widehat{=}} \\
\quad \mathit{CSST\_input\_confirm\_switch} \land \\
\quad \mathit{CSST\_valid\_request\_capability} \land \\
\quad \mathit{CSST\_in\_next\_request\_input\_state} \land \\
\quad \lnot\ \mathit{CSST\_outstanding\_fee\_condition} \land \\
\quad \lnot\ \mathit{InSTTSchedule} \land \\
\quad \mathit{StudentEnrolledInSection} \Rightarrow \\
\quad \mathit{CSST\_confirm\_switch\_feedback}
\end{array}
$$

If any one of the above conditions is not true, Telereg shall not confirm the section switch.

$$
\begin{array}{l}
\mathit{CSST\_Requirement2} \mathrel{\widehat{=}} \\
\quad \mathit{CSST\_input\_confirm\_switch} \land \\
\quad \lnot\ \mathit{CSST\_valid\_request\_capability} \lor \\
\quad \lnot\ \mathit{CSST\_in\_next\_request\_input\_state} \lor \\
\quad \mathit{CSST\_outstanding\_fee\_condition} \lor \\
\quad \mathit{InSTTSchedule} \lor \\
\quad \lnot\ \mathit{StudentEnrolledInSection} \Rightarrow \\
\quad \mathit{CSST\_confirm\_switch\_rejection}
\end{array}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{array}{l}
\mathit{Confirm\_A\_Section\_Switch\_Thread} \mathrel{\widehat{=}} \\
\quad \mathit{CSST\_Requirement1} \land \\
\quad \mathit{CSST\_Requirement2}
\end{array}
$$

Change Section Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to change a section in the student's schedule.

Stimuli

$CST\_input\_change\_section$

$Telereg\_State$

$section : Section$

$input? : Stimuli\_Or\_Responses$

$input? = change\_req(section)$

Responses

$CST\_change\_section\_feedback$

$\Delta\, Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = change\_msg$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{CST\_change\_section\_rejection}}\\
\Delta \textit{Telereg\_State}\\
\textit{output}! : \textit{Stimuli\_Or\_Responses}\\
\hline
\textit{output}! = \textit{cannot\_change\_msg}\\
\hline
\end{array}$$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$\textit{CST\_in\_next\_request\_input\_state} \mathrel{\widehat{=}}$$
$$[\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_next\_request\_input}]$$

The registration request is valid.

$$\textit{CST\_valid\_request\_capability} \mathrel{\widehat{=}} \textit{Check\_Validity\_Of\_Requests\_Capability}$$

A section corresponding to the same course as the proposed section to be switched to can be found in the student's schedule.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{CST\_old\_section\_found\_condition}}\\
\textit{Telereg\_State}\\
\textit{section} : \textit{Section}\\
\textit{id} : \textit{SId}\\
\hline
\exists\, \textit{old\_section} : \textit{Section} \bullet (\textit{CourseSchedule old\_section} = \textit{CourseSchedule section})\\
\quad \wedge (\textit{old\_section} \in \textit{RegisteredIn id})\\
\hline
\end{array}$$

The student has an outstanding fee balance.

$$\textit{CST\_outstanding\_fee\_condition} \mathrel{\widehat{=}} \textit{COFAC\_outstanding\_fee\_condition}$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the request is valid and the student does not have an outstanding fee balance, and the student is not registered in a Standard TimeTable and there are openings in the new section and the old section can be found in the student's schedule and the student is not already registered in the section, then the old section is changed to the new section in the student's schedule. Besides, the number of openings in the old section is incremented by one while that of the new section is decremented by one. The student is added to the classlist of the new section and removed from that of the old section. The student is informed of the results of the execution of this request.

$$
\begin{aligned}
&\mathit{CST\_Requirement1} \mathrel{\widehat=} \\
&\quad \mathit{CST\_input\_change\_section} \land \\
&\quad \mathit{CST\_valid\_request\_capability} \land \\
&\quad \mathit{CST\_in\_next\_request\_input\_state} \land \\
&\quad \neg\ \mathit{CST\_outstanding\_fee\_condition} \land \\
&\quad \neg\ \mathit{InSTTSchedule} \land \\
&\quad \mathit{ThereAreOpenings} \land \\
&\quad \mathit{CST\_old\_section\_found\_condition} \land \\
&\quad \neg\ \mathit{StudentEnrolledInSection} \Rightarrow \\
&\quad \mathit{RegisterStudentInSection} \land \\
&\quad \mathit{WithdrawStudentFromSection}[\mathit{old\_section}/\mathit{section}] \land \\
&\quad \mathit{CST\_change\_section\_feedback}
\end{aligned}
$$

If any one of the above conditions is not true, the student is informed that the section switch is not feasible.

$CST\_Requirement2 \mathrel{\widehat=}$

$\quad CST\_input\_change\_section \land$

$\quad \neg\ CST\_valid\_request\_capability \lor$

$\quad \neg\ CST\_in\_next\_request\_input\_state \lor$

$\quad CST\_outstanding\_fee\_condition \lor$

$\quad InSTTSchedule \lor$

$\quad \neg\ ThereAreOpenings \lor$

$\quad \neg\ CST\_old\_section\_found\_condition \lor$

$\quad StudentEnrolledInSection \Rightarrow$

$\quad CST\_change\_section\_rejection$

The specification of this thread is the conjunction of all of the above requirements:

$Change\_Section\_Thread \mathrel{\widehat=}$

$\quad CST\_Requirement1 \land$

$\quad CST\_Requirement2$

Drop Section Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to drop a section from the student's schedule.

Stimuli

DST_input_drop_section

$Telereg\_State$

$section : Section$

$input? : Stimuli\_Or\_Responses$

$input? = drop\_req(section)$

Responses

DST_drop_section_feedback

$\Delta Telereg\_State$

$output! : Stimuli\_Or\_Responses$

$output! = drop\_msg$

$$\begin{array}{|l}
\hline
\textit{DST\_drop\_section\_rejection} \\
\Delta Telereg\_State \\
output! : Stimuli\_Or\_Responses \\
\hline
output! = cannot\_drop\_msg \\
\hline
\end{array}$$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$DST\_in\_next\_request\_input\_state \mathrel{\widehat{=}}$$
$$[Telereg\_State \mid state = Waiting\_for\_next\_request\_input]$$

The registration request is valid.

$$DST\_valid\_request\_capability \mathrel{\widehat{=}} Check\_Validity\_Of\_Requests\_Capability$$

The student has an outstanding fee balance.

$$DST\_outstanding\_fee\_condition \mathrel{\widehat{=}} COFAC\_outstanding\_fee\_condition$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the request is valid and the student does not have an outstanding fee balance and the student is not registered in a Standard TimeTable and the section can be found in the student's schedule, then it is deleted and the student is removed from the classlist associated with the section. The number of openings in that section is incremented by one. The student is informed that the section has been dropped.

$$
\begin{aligned}
&\mathit{DST\_Requirement1} \mathrel{\widehat{=}} \\
&\quad \mathit{DST\_input\_drop\_section} \land \\
&\quad \mathit{DST\_valid\_request\_capability} \land \\
&\quad \mathit{DST\_in\_next\_request\_input\_state} \land \\
&\quad \neg\ \mathit{DST\_outstanding\_fee\_condition} \land \\
&\quad \neg\ \mathit{InSTTSchedule} \land \\
&\quad \mathit{StudentEnrolledInSection} \Rightarrow \\
&\quad \mathit{WithdrawStudentFromSection} \land \\
&\quad \mathit{DST\_drop\_section\_feedback}
\end{aligned}
$$

If any one of the above conditions is not true, the student is informed that the section has not been dropped from the student's schedule.

$$
\begin{aligned}
&\mathit{DST\_Requirement2} \mathrel{\widehat{=}} \\
&\quad \mathit{DST\_input\_drop\_section} \lor \\
&\quad \neg\ \mathit{DST\_valid\_request\_capability} \lor \\
&\quad \neg\ \mathit{DST\_in\_next\_request\_input\_state} \lor \\
&\quad \mathit{DST\_outstanding\_fee\_condition} \lor \\
&\quad \mathit{InSTTSchedule} \lor \\
&\quad \neg\ \mathit{StudentEnrolledInSection} \Rightarrow \\
&\quad \mathit{DST\_drop\_section\_rejection}
\end{aligned}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{aligned}
&\mathit{Drop\_Section\_Thread} \mathrel{\widehat{=}} \\
&\quad \mathit{DST\_Requirement1} \land \\
&\quad \mathit{DST\_Requirement2}
\end{aligned}
$$

Look Up Inquiry On Sections Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.

Overview

This thread describes the processing performed upon receiving a request to look up the number of openings in a section.

Stimuli

$LOIST\_input\_lookup\_section$

$Telereg\_State$

$section : Section$

$input? : Stimuli\_Or\_Responses$

$input? = lookup\_req(section)$

Responses

$LOIST\_lookup\_section\_feedback$

$\Delta Telereg\_State$

$section : Section$

$output! : Stimuli\_Or\_Responses$

$output! = lookup\_msg(Availability\ section)$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$LOIST\_in\_next\_request\_input\_state \mathrel{\widehat=}$$
$$[\,Telereg\_State \mid state = Waiting\_for\_next\_request\_input]$$

The registration request is valid.

$$LOIST\_valid\_request\_capability \mathrel{\widehat=} Check\_Validity\_Of\_Requests\_Capability$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg shall validate the request. If the request is valid, Telereg shall report the number of openings for the section.

$$\begin{aligned}&Look\_Up\_Inquiry\_On\_Sections\_Thread \mathrel{\widehat=}\\ &\quad LOIST\_input\_lookup\_section \land\\ &\quad LOIST\_valid\_request\_capability \land\\ &\quad LOIST\_in\_next\_request\_input\_state \Rightarrow\\ &\quad LOIST\_lookup\_section\_feedback\end{aligned}$$

List All Courses Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to list all the courses in the student's schedule.

Stimuli

LACT_input_list_all

$Telereg\_State$

$input? : Stimuli\_Or\_Responses$

$input? = list\_req$

Responses

LACT_list_all_feedback

$\Delta Telereg\_State$

$id : SId$

$output! : Stimuli\_Or\_Responses$

$\forall s : Section \bullet s \in (RegisteredIn\ id) \land output! = list\_msg(CourseSchedule\ s)$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$LACT\_in\_next\_request\_input\_state \mathrel{\widehat{=}}$$
$$[Telereg\_State \mid state = Waiting\_for\_next\_request\_input]$$

The registration request is valid.

$$LACT\_valid\_request\_capability \mathrel{\widehat{=}} Check\_Validity\_Of\_Requests\_Capability$$

The student has an outstanding fee balance.

$$LACT\_outstanding\_fee\_condition \mathrel{\widehat{=}} COFAC\_outstanding\_fee\_condition$$

Actions

None.

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the request is valid and the student does not have an outstanding fee balance, Telereg shall return the list of sections in the student's schedule.

$$\begin{aligned}
&List\_All\_Courses\_Thread \mathrel{\widehat{=}}\\
&\quad LACT\_input\_list\_all \land\\
&\quad LACT\_valid\_request\_capability \land\\
&\quad LACT\_in\_next\_request\_input\_state \land\\
&\quad \lnot\ LACT\_outstanding\_fee\_condition \Rightarrow\\
&\quad LACT\_list\_all\_feedback
\end{aligned}$$

Cancel Entire Registration Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Dependencies

1. Check_Validity_Of_Requests_Capability as specified in \Check Validity Of Requests Capability\.
2. COFAC_outstanding_fee_condition as specified in \Check Outstanding Fee Assessment Condition\.

Overview

This thread describes the processing performed upon receiving a request to cancel the student's entire registration.

Stimuli

```
┌─ CERT_input_cancel_all ─────────────────────────
│ Telereg_State
│ input? : Stimuli_Or_Responses
├──────────
│ input? = cancel_req
└─────────────────────────────────────────────────
```

Responses

```
┌─ CERT_cancel_all_feedback ──────────────────────
│ ΔTelereg_State
│ output! : Stimuli_Or_Responses
├──────────
│ output! = cancel_msg
└─────────────────────────────────────────────────
```

$$\begin{array}{|l}
\hline
\textit{CERT\_cancel\_all\_rejection} \\
\Delta \textit{Telereg\_State} \\
\textit{output!} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{output!} = \textit{cancel\_rejection} \\
\hline
\end{array}$$

Conditions and Modes

Telereg is in "Waiting_for_next_request_input" state.

$$\textit{CERT\_in\_next\_request\_input\_state} \mathrel{\widehat=}$$
$$[\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_next\_request\_input}]$$

The registration request is valid.

$$\textit{CERT\_valid\_request\_capability} \mathrel{\widehat=} \textit{Check\_Validity\_Of\_Requests\_Capability}$$

The student has an outstanding fee balance.

$$\textit{CERT\_outstanding\_fee\_condition} \mathrel{\widehat=} \textit{COFAC\_outstanding\_fee\_condition}$$

Actions

Each section in the student's schedule is removed from the schedule and the student is removed from the class list associated with each section. Also, the number of openings in each section is incremented by one.

$$
\begin{array}{|l}
\textit{CERT\_remove\_each\_section} \\
\hline
\Delta \textit{Telereg\_State} \\
id : SId \\
newclasslist : Classlist \\
newopenings : Openings \\
newschedule : Schedule \\
\hline
\forall section : Section \bullet \\
\qquad StudentEnrolledInSection \land WithdrawStudentFromSection \\
\hline
\end{array}
$$

Requirements

In the "Waiting_for_next_request_input" state, Telereg will validate the request. If the request is valid and the student does not have an outstanding fee balance and the student is not registered under the Standard TimeTable, then for each of the sections that the student has registered for, it is deleted from the student's schedule. The student is removed from the classlist associated with each section and the number of openings in each section is incremented by one. Feedback of success of operation is given to the student.

$$
\begin{array}{l}
CERT\_Requirement1 \mathrel{\widehat{=}} \\
\qquad CERT\_input\_cancel\_all \land \\
\qquad CERT\_valid\_request\_capability \land \\
\qquad CERT\_in\_next\_request\_input\_state \land \\
\qquad \neg\ CERT\_outstanding\_fee\_condition \land \\
\qquad \neg\ InSTTSchedule \Rightarrow \\
\qquad CERT\_remove\_each\_section \land \\
\qquad CERT\_cancel\_all\_feedback
\end{array}
$$

If any one of the above conditions is not true, then the student is informed that her request to cancel her registration cannot be done.

$$
\begin{array}{l}
CERT\_Requirement2 \mathrel{\widehat=} \\
\qquad CERT\_input\_cancel\_all \land \\
\qquad \lnot\ CERT\_valid\_request\_capability \lor \\
\qquad \lnot\ CERT\_in\_next\_request\_input\_state \lor \\
\qquad CERT\_outstanding\_fee\_condition \lor \\
\qquad InSTTSchedule \Rightarrow \\
\qquad CERT\_cancel\_all\_rejection
\end{array}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{array}{l}
Cancel\_Entire\_Registration\_Thread \mathrel{\widehat=} \\
\qquad CERT\_Requirement1 \land \\
\qquad CERT\_Requirement2
\end{array}
$$

Add Specialization Thread

History

Created 7 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 13 March 1994 by H.Wong; changes made to the SIS-Telereg system spec.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.
Modified 27 July 1994 by H.Wong.

Dependencies

None.

Overview

This thread describes the processing performed upon receiving a request to add a specialization to the student's record.

Stimuli

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AST\_input\_add\_specialization}} \\
\hline
Telereg\_State \\
spec : Specialization \\
input? : Stimuli\_Or\_Responses \\
\hline
input? = add\_spec\_req(spec) \\
\hline
\end{array}$$

Responses

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AST\_add\_specialization\_feedback}} \\
\hline
\Delta Telereg\_State \\
id : SId \\
output! : Stimuli\_Or\_Responses \\
\hline
output! = add\_spec\_msg(StudentSpec\ id) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AST\_invalid\_specialization\_rejection}} \\
\hline
\Delta \textit{Telereg\_State} \\
\textit{output}! : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{output}! = \textit{invalid\_spec\_msg} \\
\hline
\end{array}$$

Conditions and Modes

Telereg is in “Waiting_for_next_request_input” state.

$$\textit{AST\_in\_next\_request\_input\_state} \mathrel{\widehat=} \\ [\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_next\_request\_input}]$$

The specialization is valid.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AST\_valid\_spec\_condition}} \\
\hline
\textit{Telereg\_State} \\
\textit{spec} : \textit{Specialization} \\
\hline
\textit{spec} \in \textit{ValidSpec} \\
\hline
\end{array}$$

Telereg is in “Waiting_for_spec_input” state.

$$\textit{AST\_in\_spec\_input\_state} \mathrel{\widehat=} \\ [\textit{Telereg\_State} \mid \textit{state} = \textit{Waiting\_for\_spec\_input}]$$

Actions

$$\textit{AST\_move\_to\_next\_request\_input\_state} \mathrel{\widehat=} \\ [\Delta \textit{Telereg\_State} \mid \textit{state}' = \textit{Waiting\_for\_next\_request\_input}]$$

Telereg shall add the specialization to the student’s record.

$$\begin{array}{|l}
\textit{AST\_add\_spec\_to\_record} \\
\hline
\Delta \textit{Telereg\_State} \\
\textit{id} : \textit{SId} \\
\textit{spec} : \textit{Specialization} \\
\hline
\textit{StudentSpec}' = \textit{StudentSpec} \oplus \{\textit{id} \mapsto \textit{spec}\} \\
\hline
\end{array}$$

Requirements

In the "Waiting_for_next_request_input" state or "Waiting_for_spec_input" state, Telereg will validate the specialization. If the specialization is not valid, the student is informed of it. Telereg will stay in its initial state.

$$\begin{array}{l}
\textit{AST\_Requirement1} \mathrel{\widehat{=}} \\
\quad \textit{AST\_input\_add\_specialization} \land \\
\quad \textit{AST\_in\_next\_request\_input\_state} \lor \\
\quad \textit{AST\_in\_spec\_input\_state} \land \\
\quad \neg\ \textit{AST\_valid\_spec\_condition} \Rightarrow \\
\quad \textit{AST\_invalid\_specialization\_rejection}
\end{array}$$

In the "Waiting_for_next_request_input" state, Telereg will validate the specialization. If the specialization is valid, Telereg shall add the specialization to the student's record and give as feedback the specialization on file for the student.

$$\begin{array}{l}
\textit{AST\_Requirement2} \mathrel{\widehat{=}} \\
\quad \textit{AST\_input\_add\_specialization} \land \\
\quad \textit{AST\_in\_next\_request\_input\_state} \land \\
\quad \textit{AST\_valid\_spec\_condition} \Rightarrow \\
\quad \textit{AST\_add\_spec\_to\_record} \land \\
\quad \textit{AST\_add\_specialization\_feedback}
\end{array}$$

In the "Waiting_for_spec_input" state, Telereg will validate the specialization. If the specialization is valid, Telereg shall add the specialization to the student's record and

$$
\begin{array}{l}
Handle\_Hang\_Up\_Process \mathrel{\widehat=} \\
\quad HHUP\_hangup\_condition \Rightarrow \\
\quad HHUP\_move\_to\_idle\_state
\end{array}
$$

give as feedback the specialization on file for the student. Telereg shall commit to "Waiting_for_next_request_input" state.

$$
\begin{aligned}
&AST\_Requirement3 \mathrel{\widehat{=}} \\
&\quad AST\_input\_add\_specialization \land \\
&\quad AST\_in\_spec\_input\_state \land \\
&\quad AST\_valid\_spec\_condition \Rightarrow \\
&\quad AST\_add\_spec\_to\_record \land \\
&\quad AST\_add\_specialization\_feedback \land \\
&\quad AST\_move\_to\_next\_request\_input\_state
\end{aligned}
$$

The specification of this thread is the conjunction of all of the above requirements:

$$
\begin{aligned}
&Add\_Specialization\_Thread \mathrel{\widehat{=}} \\
&\quad AST\_Requirement1 \land \\
&\quad AST\_Requirement2 \land \\
&\quad AST\_Requirement3
\end{aligned}
$$

Handle Hang Up Process

History

Created 13 March 1994 by H.Wong; based on the threads-based natural language specification of the SIS-Telereg system.
Updated 16 March 1994 by H.Wong; changes made to the SIS-Telereg system spec, draft 1.
Modified 7 April 1994 by H.Wong; typechecked with fuzz.

Dependencies

None.

Overview

This process describes the processing performed upon detection of a hang up.

Conditions and Modes

Telereg has detected a hang up.

$$\begin{array}{|l}
\textit{HHUP\_hangup\_condition} \\
\hline
\textit{Telereg\_State} \\
\textit{input?} : \textit{Stimuli\_Or\_Responses} \\
\hline
\textit{input?} = \textit{hangup} \\
\hline
\end{array}$$

Actions

Telereg shall commit to "IDLE" state.

$$\textit{HHUP\_move\_to\_idle\_state} \mathrel{\widehat{=}} [\Delta \textit{Telereg\_State} \mid \textit{state}' = \textit{IDLE}]$$

Requirements

In any state that Telereg may be in, when Telereg detects a hang up, Telereg shall complete the transaction it was executing if it is in the midst of executing a transaction. Telereg shall commit to the "IDLE" state.