THE UNIVERSITY OF CALGARY

Integrating Informal and Formal Requirements Methods:

A Practical Approach for Systems Employing Spatially Referenced Data

by

Alan D. Goodbrand

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

OCTOBER, 2000

ABSTRACT

From any Requirements Specification, the intention is to develop a workable system. It is therefore the responsibility of the Requirements Engineer to state the requirements in such a way as to minimize ambiguity and increase precision.

Formal methods were developed to express requirements which were not open to interpretation (Hayes, 1987). They are best used in areas that suit expression via mathematics (Barden et al., 1994). Spatially referenced data seems well suited to this type of specification due to the fact that it is represented by sets. Formal methods are best utilized in the areas of set acquisition, manipulation, retrieval and representation (Barden et al., 1994).

This thesis sets out to show that a Requirements Specification for a system using Spatially Referenced data can benefit in the areas of precision and reduced ambiguity with the integration of formal methods. Changes made to the Informal Requirements as a result of their formal specification will be used to show this.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

1.0 Introduction

The Institute of Electrical and Electronics Engineers (IEEE) defines a requirement as (Macaulay, 1996):

1. "A condition or capacity needed by a user to solve a problem or achieve an objective."
2. "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents."
3. "A documented representation of a condition or capability as in 1 or 2." (p 4)

This definition is correct at the very highest level of requirements gathering but what must be remembered is that with a Requirements Specification, the intention is to develop a workable system. Therefore it is the responsibility of the Requirements Engineer to state the requirements in such a way as to minimize ambiguity and be as precise as possible. The majority of system development projects that fail, do so due to insufficient requirements specification or ambiguous specifications (Gause & Weinberg, 1989).

When specifications are written in a natural language, such as English, they tend to have contradictions, ambiguities and omissions. The solution to this is to express specifications formally using mathematical terminology (Schach, 1993).

1.1 Formal Requirements Specifications

Formal requirements specifications were developed in an effort to express requirements in a precise mathematical language which was not open to interpretation (Hayes, 1987). Using

this method. if the Requirements Engineer expressed the requirements using mathematics and the system developer understood that mathematics, the system that resulted would accurately satisfy the requirements.

Formal requirements therefore result in a more precise system development phase than that which would normally be obtained if the requirements were expressed in informal plain language. Formal requirements should be used, however, only in areas that best suit expression via mathematical constructs (Barden et al., 1994). It is finding that dividing line which is most important. How can we properly integrate informal and formal requirements acquisition methods? This question cannot be answered without first examining the data. In other words, there is no correct answer to this question of integrating informal and formal requirements methods. The answer depends on the type of data with which you are dealing. This thesis chooses to deal with the aspect of integrating informal and formal requirements acquisition methods within the context of spatially referenced data. That is, data that is or can be represented on maps. This particular type of data seems best suited to formal requirements due to the fact that it is best represented by sets. Formal requirements are best utilized in the areas of set acquisition, manipulation, retrieval and representation (Barden et al., 1994) and spatial data is an area that may benefit from this method.

This thesis sets out a framework within which formal requirements methods can be used in systems that utilize spatially referenced data. It's specific context is within a Land Management Information System (LMIS) utilized within the province of Alberta, Canada using the Dominion Land Survey system of representation. A complete informal specification, written by the author in his professional capacity as an Information

Technology consultant, is included in Appendix A of this document for just such a system. The methods utilized within are therefore specific to that particular frame of reference. Extrapolation of the findings of this thesis into broader areas of research and usage is beyond its scope and can be an area for future research.

1.2 Domain Background

The province of Alberta, Canada has a predominantly energy based economy, mainly oil and gas. The nature of these products are that they must be extracted from the ground in place and shipped elsewhere for refinement and market. The majority of this product transportation takes place via underground pipelines crisscrossing the province, of which there are approximately 23,000 kilometers within Alberta alone. The owners of these pipelines must keep track of all pertinent data with regards to each and every kilometer of pipe that they own as well as all pertinent data in regards to the land where the pipeline resides. To accomplish this task, the major pipeline companies each use a "Land Management Information System" (LMIS) of one sort or another.

The nature of the data involved, plus the fact that the province of Alberta uses the Dominion Land Survey nomenclature for land title description, makes this sort of product highly set oriented. These sets are related by relations and functions which can be precisely defined. It is these factors that lead the author to believe that this type of application is one which can benefit from the use of formal requirements specification methods.

1.3 Aims and Objectives

The aim of this thesis is therefore to show that when a formal specification is incorporated into the Requirements Specification Document for an LMIS system, the specification that results will be of greater precision.

In order to accomplish this aim, the following objectives must be met:

1. Understand the current state of the art – The current state of the art for this thesis consists of three separate areas, namely: Requirements Engineering, Spatial Data Systems and Formal Methods. Evaluate each area separately followed by material dealing with their integration.
2. Develop an informal Requirements Specification for a Land Management Information System.
3. Develop a formal requirements specification for systems employing spatially referenced data. This will be achieved by taking the informal Requirements Specification Document developed as the second objective and then developing a formal specification for the same system. An existing utilization of Formal Methods for a system of this nature could not be found in the literature.
4. Critical analysis of the method - From the above objective, a complete specification for an LMIS system will result. How do we know that this is a better specification then one produced by informal methods? This objective will be met by tracking changes to the informal specification that have taken place due to it being specified formally. These changes to the informal specification will be the result of

discovered errors or ambiguity which were brought to light as a result of the formal specifications.

With the completion of these objectives it is intended that the existing Software Engineering body of knowledge will be enriched through the addition of this case study demonstrating the applicability and practicality of Formal Methods. The formal specification of software is not widely utilized in industry in North America due mainly to a lack of available information and examples for the professional Requirements Engineer. This thesis, with its aims and objectives, researches the area of the applicability and practicality of Formal Methods in the specific context of systems employing spatially referenced data and sets out to aid the professional Requirements Engineer in his continuing quest for better tools and methods.

1.4 Informal Specification Background

Appendix A of this thesis contains a complete informal specification for a Land Management Information System. This informal specification satisfies objective 2 as stated above and is used to develop the Formal Specification required by objective 3. It uses a narrative format, as prescribed by the client, and approximately follows the IEEE Guide to Software Requirements Specifications (IEEE, 1984).

The product being specified provides information about parcels of land and allows land management activities. It must be available 24 hours a day, 7 days a week. It must provide an on-line, visual representation of all land parcels within the province of Alberta on which the company has an obligation. By obligation, it is meant that the company either owns the

land, leases the land, has a right-of-way through the land or has an obligation to the landowner or occupant due to having other obligations on adjacent parcels of land.

The requirements for this system will be described from the perspective of the 5 user groups, namely:

Construction – This group deals with land issues prior to construction of a pipeline or other facility. Their function is to determine landowners and regulations pertaining to the affected parcels of land. They must then complete all legal requirements to confirm that construction can begin on the affected land at the required time.

Appraisals – When a right-of-way is required through a parcel of land, obligations are incurred by the company. These obligations are usually, but not always, in the form of monetary compensation. It is the job of the Appraisals group to determine the value of such compensation.

Operations – This user group deals with the day to day issues of land after construction has been completed. This includes, completing all contractual obligations for the ongoing use of the land, maintaining accurate information in regards to the land and responding to all inquiries with regards to additional uses of the land.

Land Agents – Land Agents are the people that communicate directly with the landowners. They gather information, resolve disputes and generally maintain a good rapport between the landowner or occupant and the Company.

Legal – The legal department ensures that all regulations and obligations between the landowner, company and government are properly completed.

1.5 Thesis Structure

Chapter 2 describes the Current State of the Art. It consists of background material with regards to representing spatial data in Alberta as well as a review of the current state of Requirements Engineering with particular emphasis to spatially referenced data and the integration of formal and informal methods. The product "Mapguide" from Autodesk, the "Z" specification language and the "Z" specification software "Formaliser" from Logica UK are also described.

Chapter 3 is a complete Formal Specification for the Land Management Information System described as an informal narrative specification in Appendix A.

Chapter 4 is an evaluation of the completed Formal Specification in Chapter 3 to determine what it has actually accomplished, it's practicality and usefulness.

Chapter 5 concludes the thesis. It concentrates on how this research satisfied the aim and objectives and what additional areas there are for further research.

Appendix A consists of a complete Requirements Specification Document entitled "Requirements for a Land Management Information System". This document was created by the author of this thesis in his professional capacity as an Information Systems consultant and is used as the basis for the creation of the Formal Specification. It uses an informal narrative format, as prescribed by the client, and was created through interviews with stakeholders from each of the various user groups.

Chapter 2

2.0 Current State of the Art

This chapter presents an overview of the relevant literature found available for this Thesis. This chapter has been divided into three separate categories: Requirements Engineering, Spatial Data Systems and Formal Methods. Each of these disciplines is mature in its own right, but very little has been done to specify spatial data systems formally. Formal methods have not been employed before in the Requirement stage of a Spatial Data project that the author is aware of.

2.1 Requirements Engineering

The Institute of Electrical and Electronics Engineers (IEEE) defines the term *requirement* as (Macaulay, 1996):

1. "A condition or capacity needed by a user to solve a problem or achieve an objective."
2. "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents."
3. "A documented representation of a condition or capability as in 1 or 2." (p 4)

Requirements Engineering can be defined as the systematic process of developing requirements through an iterative co-operative process of analyzing the problem,

documenting the resulting observations and checking the accuracy of the understanding gained (Pohl, 1993).

The intention of Requirements Engineering is therefore the eliciting and documenting of system and stakeholder requirements for the purpose of developing a workable system. It is therefore the responsibility of the Requirements Engineer to state the requirements in such a way as to minimize ambiguity and be as precise as possible. The majority of system development projects that fail do so due to insufficient requirements specifications or ambiguous specifications (Gause & Weinberg, 1989).

The concepts of what a requirement is and various ways of eliciting them and documenting them are well described in the RE literature. Various approaches, including Structured Analysis, Joint Application Design, Soft Systems Methodology, Interactive and Group Session approaches are well covered in the literature (Macaulay, 1996). The essential role of the requirements phase is to ensure that the users' needs are properly understood before designing and implementing a system to meet them (Humphrey, 1990). In all cases the end product of the successful Requirements Engineering process is a "Requirements Specification Document (RSD)." (Macaulay, 1996).

The particular method used for the creation of the informal RSD section of this thesis is that of a modified "Structured Analysis". This method was not chosen by the author but was at the direction of the client for whom this RSD was prepared. This technique involves the interviewing of users and collecting and analyzing information from various documents (Macaulay, 1996). Dataflow Diagrams and Entity Relationship Diagrams are two of the

main tools used by the Requirements Engineer practicing Structured Analysis (Buckley, 1989).

Dataflow Diagrams are used to depict data and process (Woodman, 1990). They depict how and what data enters a process, how and what data leaves a process and the transformation process itself. It is a simple style which has both strengths and weaknesses. Creating Dataflow Diagrams can be learned easily and they are very easy to understand, but that simplicity implies a lack of sophistication which can lead to a lack of expressive power and disagreements in interpretation (Woodman, 1990). Dataflow Diagrams were originally introduced between 1977 and 1979 by Gane and Sarson (1979), Yourdon and Constantine (1978) and DeMarco (1978) with each successive work building upon the work of the other. Figure 2.1 is a small sample Dataflow Diagram which illustrates a basic process, that of producing a simple payroll.

In this specific Dataflow Diagram, data is recorded on a Timecard by a 'Time Keeping Clerk' and flows to the 'Calculate Pay' process. The 'Pay Calculations' flow to the 'Prepare Paycheck' process. As can be seen by this particular example, processes have inputs and outputs and are designated by rectangles with rounded corners. A horizontal line separates the process designation from its description. Sources or stores of data, be they people or file storage, are designated by squares. The direction of data flow is shown by a directed line segment with the method of data flow listed along side the arrow.

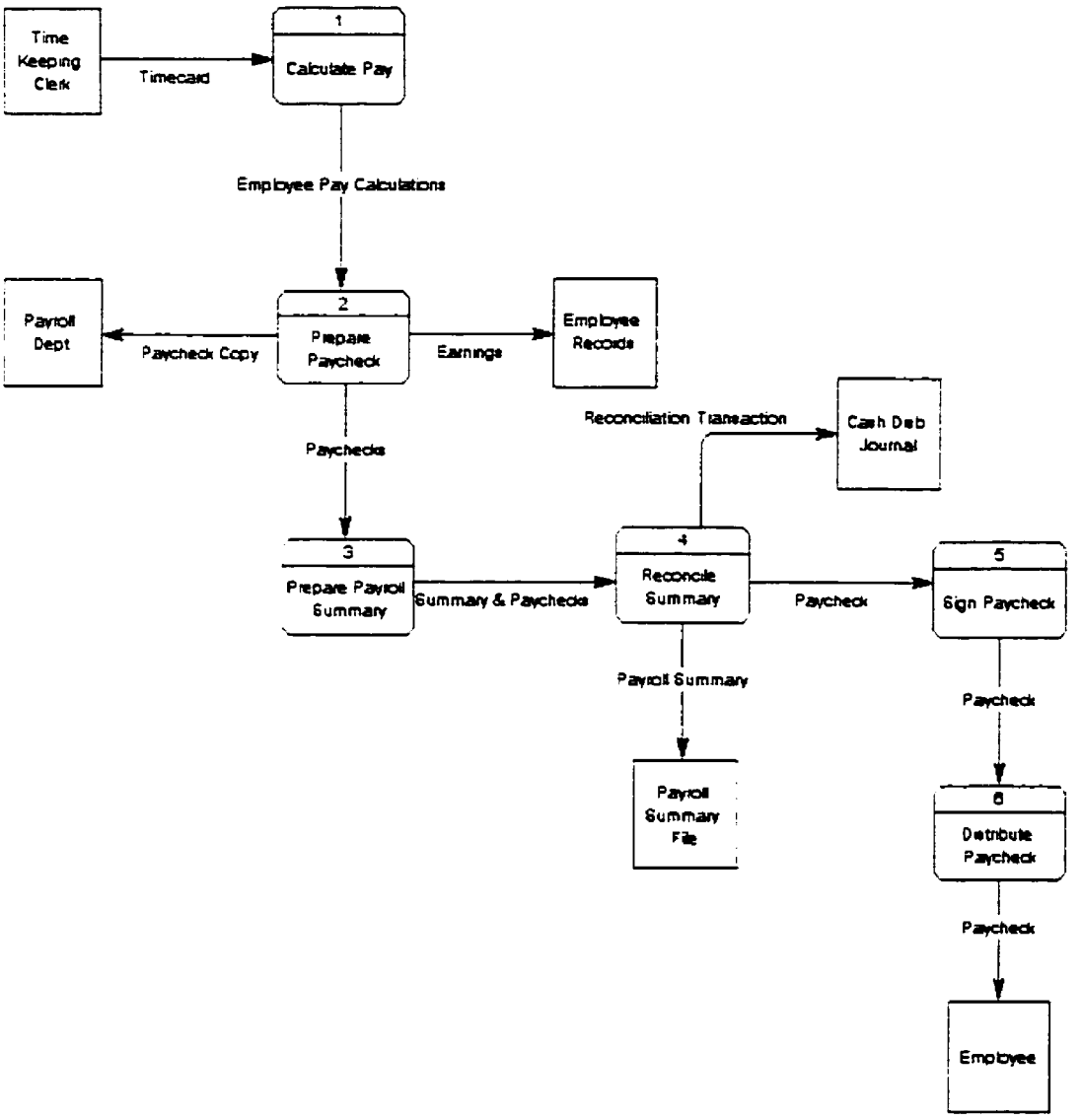


Figure 2.1 - Dataflow Diagram
(http://pink1.bschool.ukans.edu/Administration/dataflow.htm)

Entity Relationship Diagrams are similar to Dataflow Diagrams in that they graphically relay information. Whereas Dataflow Diagrams show input, process and output, ER diagrams show relationships and information about relationships. Relationships may be one

to one. one to many. many to one. many to many etc. An example is shown below as figure 2.2.

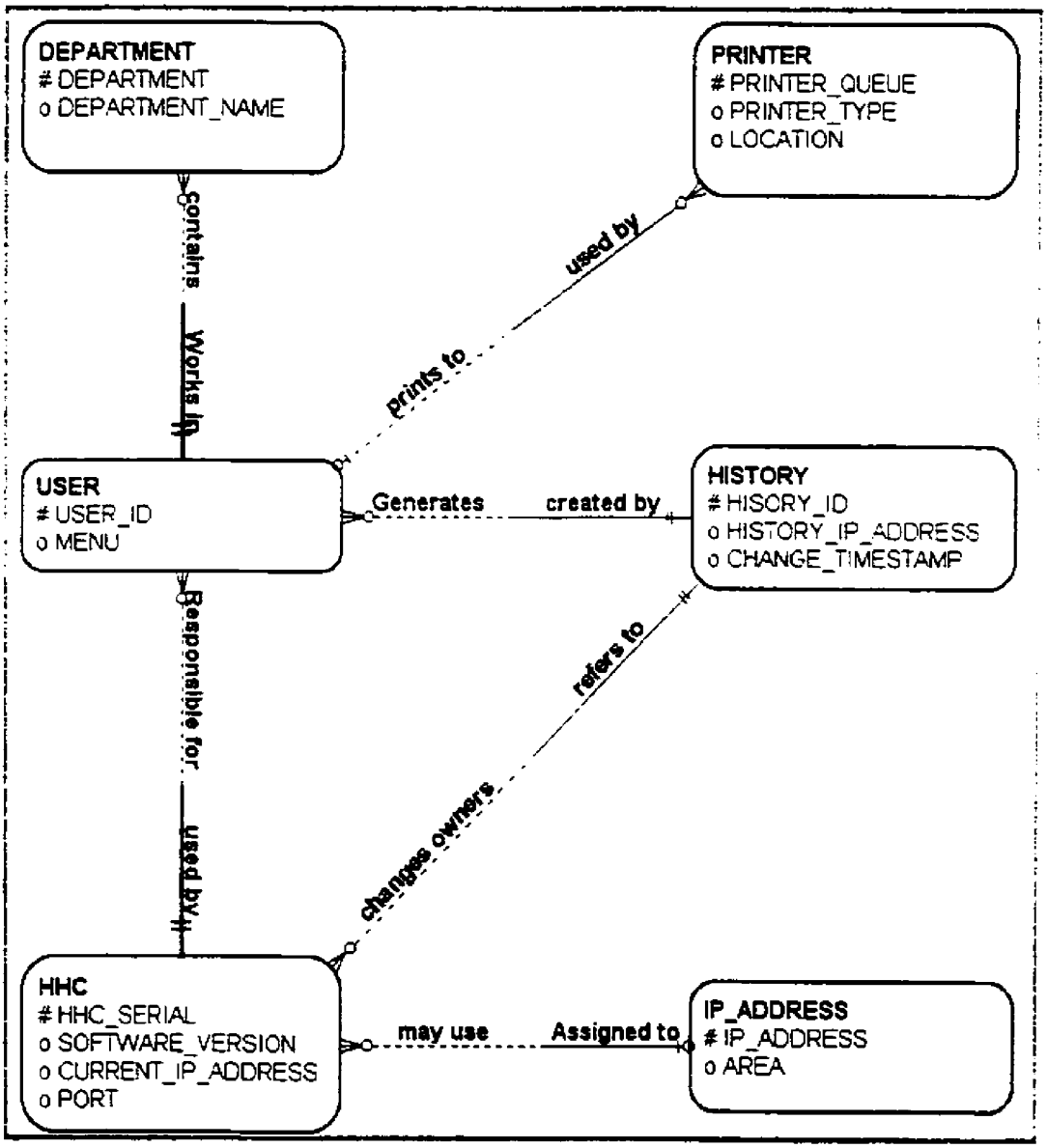


Figure 2.2 – Entity Relationship Diagram

(http://www.scis.nova.edu/~hunterth/MCIS630/week04.html)

Descriptions of how to create a Requirements Specification Document using a narrative technique are well documented (Macaulay, 1996) (Humphrey, 1990) (Buckley, 1989). The format for this particular RSD is defined by the client for whom this project is being done. It does, however, closely follow the IEEE Guide to Software Requirements Specifications (IEEE, 1984) being laid out as follows:

Table of Contents

In order to measure the increase in precision of the RSD with the application of formal methods, changes to the informal requirements will be tracked and change control will be practiced. Requirements Management, being a Key Process Area of the Capability Maturity

Model (Paulk, 1993), will be managed in such a way as to establish a baseline after the Informal Requirements have been produced following the Structured Analysis method. Changes to the requirements as a result of the integration of formal methods will be identified through Requirements Management and indicate a change in precision.

Most of the systems that have been specified using the Structured Analysis method do not possess the mathematical orientation that spatially referenced systems do. This, and the fact that formal methods are best used in areas of set manipulation (Barden et al., 1994) leads the author to believe that a Requirements Specification Document for a system employing spatially referenced data and the resulting system will benefit from the application of formal methods.

2.2 Spatial Data Systems

Spatial data is data that is or can be displayed on maps. Within the province of Alberta, Canada, where the University of Calgary resides, a great deal of the industry is energy based, primarily oil and gas. These natural resources require that anyone who wants to exploit them come to them and be involved with the land. People own land. People reside on land. People hold rights to land. All of these things have to be accommodated when dealing with natural resources.

Spatial data, in the context of this thesis, refers also to the attributes connected with the land. Parcels may be of any size, anywhere in the province. Each parcel has several attributes associated with it: Location, size, owner, classification, rights held, resources

available etc. Each of these attributes must be captured, held on a database, maintained, manipulated, inquired upon intelligently etc.

The province of Alberta is divided using the Dominion Land Survey system, a system which was used initially in the 19th century. This system is described in detail in "Understanding Western Canada's Dominion Land Survey System" by Robert McKercher and Bertram Wolfe (1986). A thorough understanding of this system is required in order to understand the specific details of the requirements as stated by the Land Engineers.

While understanding the Dominion Land Survey System employed in Western Canada is not a requirement to understanding the requirements gathering techniques developed, having an understanding of the underlying conventions will assist the reader in assessing it's viability for other projects.

Western Canada's Dominion Land Survey System. (McKercher, 1989)

Grid layout

The Western Canadian system of land description known as the Dominion Land Survey (D.L.S.) system allows anyone to pinpoint any parcel of land as small as 4 hectares anywhere on the dominion grid. For the purposes of this document, the system is applicable between latitudes 49°N (the Canada U.S. border) and 60°N (the border with the NorthWest Territories) and the first meridian at 97°27'28.4"W and the coastal meridian at 122°45'39.6"W. There are 8 lines of meridian used within the system as defined below.

1. First meridian at 97°27'28.4"W longitude.
2. Second meridian at 102°W longitude.
3. Third meridian at 106°W longitude.

4. Fourth meridian at 110°W longitude.
5. Fifth meridian at 114°W longitude.
6. Sixth meridian at 118°W longitude.
7. Seventh meridian at 122°W longitude.
8. Coastal meridian at 122°45'39.6"W longitude.

These meridian lines were defined in 1869 when the first Western Canada Survey was begun. The choice of 97°27'28.4"W as the first meridian was arbitrary as it defined the western most limit of settlement up to that date. The choice of the Coastal meridian being 122°45'39.6"W. approximately 55 km west of the Seventh meridian. occurred at a later date and was due to it's initial survey being conducted by the British Columbia provincial government. not the Dominion government. and it being incorporated into the D.L.S. at a later date.

Beginning at the intersection of 97°27'28.4"W (the first meridian) and 49°N (the Canada U.S. border) and proceeding towards the north-west. a grid was established using east-west running lines called "Township lines" and north-south running lines known as "Range lines". These lines were set at 6 miles (approximately 10km) apart (See figure 2.3). (McKercher. 1989)

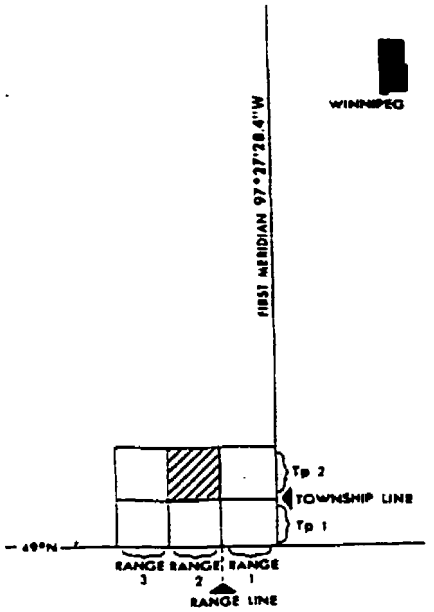


Figure 2.3 – Grid Layout

This method of dividing the land into 6 miles square townships proceeded in a westerly direction until reaching the next meridian. At that point the meridian line became the western most boundary of the township. These townships were still 6 miles from north to south but less than 6 miles from east to west. Commencing at the next meridian, the process was repeated beginning with a new 6 x 6 mile township immediately to the west of the meridian. This method of designation ran northwest from the intersection of each meridian and the 49th parallel, encompassing all habitable areas of the provinces.

Complications

The township lines running east and west all run parallel following the lines of latitude. The number of townships between the 49th parallel and the 60th parallel is therefore constant at 126. The lines of longitude however converge. If the range lines, which run north and

south, then followed the lines of longitude established at 6 miles at the 49th parallel northward, the distance between range lines would be considerably less than 6 miles at the 60th parallel. To account for this fact, the second township line was established as a correction line as well as every fourth township line north. The southern most border of the new townships were re-surveyed at 6 miles and the range lines again followed a new line of longitude. What resulted from this was that the townships immediately north of a correction line had a southern boundary of 6 miles and a northern boundary of approximately 115' (35 m) less. A township with it's northern boundary on a correction line however had a southern boundary of 345' (105 m) less and a northern boundary of 460' (140 m) less than the township immediately north of it. See figure 2.4 (McKercher, 1989) for a visual representation of this.

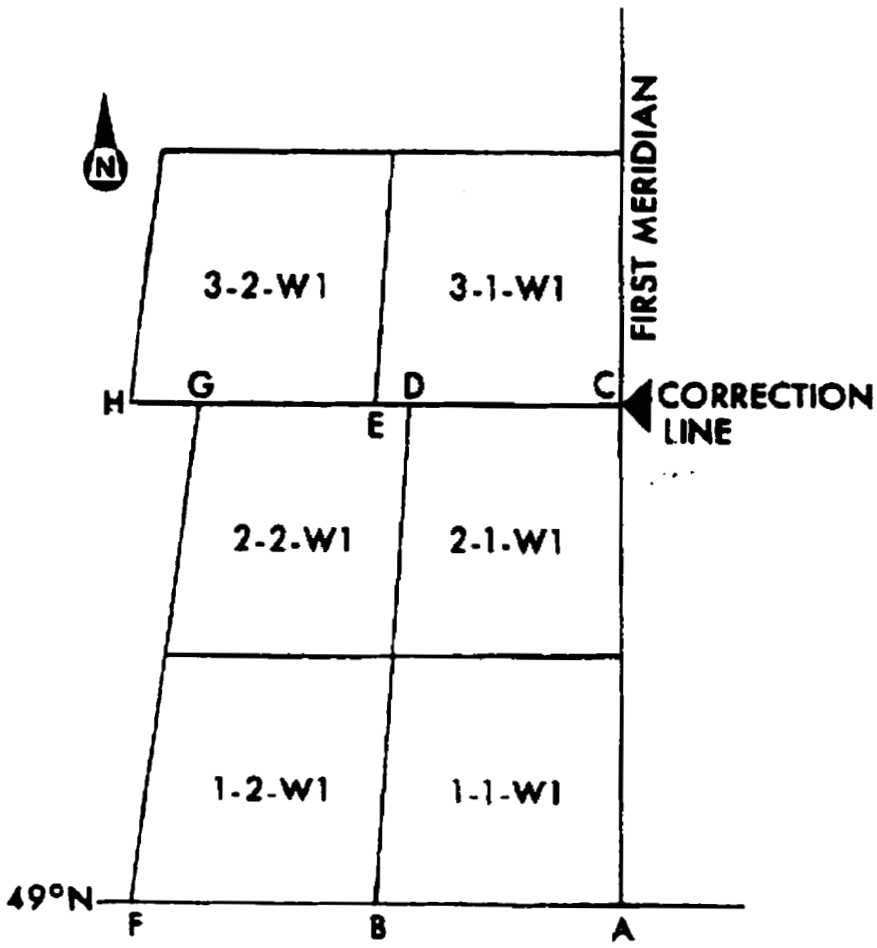


Figure 2.4 – Correction Lines

While the number of townships on a north-south line between the 49th parallel and the 60th parallel therefore is a constant 126. the number of townships on an east-west line between meridians varies from a maximum of 36 at the 49th parallel to a minimum of 23 at the 60th parallel.

Subdividing Townships

Given the grid system shown above, it is possible with only 3 numbers to locate any township in the area bordered by the 49th parallel, the 60th parallel, the first meridian and the Alberta, British Columbia border. This set of 3 numbers is by convention expressed in the order township, range, meridian. For example, 12-9 W4 is the legal description for township 12 meaning the 12th township north of the 49th parallel, range 9 west of the 4th meridian. Townships are always expressed as West of a certain meridian west of the first meridian. The system actually does carry on east of the first meridian where the grid logic is mirrored and townships are expressed as being east of a certain meridian, but that is outside the scope of this thesis.

Each township is approximately 23.040 acres (9.325 hectares) in size. Each township is therefore divided into up to 36 square sections each approximately 1 mile (1.6 km) on each side. The standard 6 mile by 6 mile township therefore becomes 36 1 mile by 1 mile sections. These sections number from the south east most corner winding in a snake like fashion until section 36 in the north-east corner (figure 2.5). (McKercher, 1989).

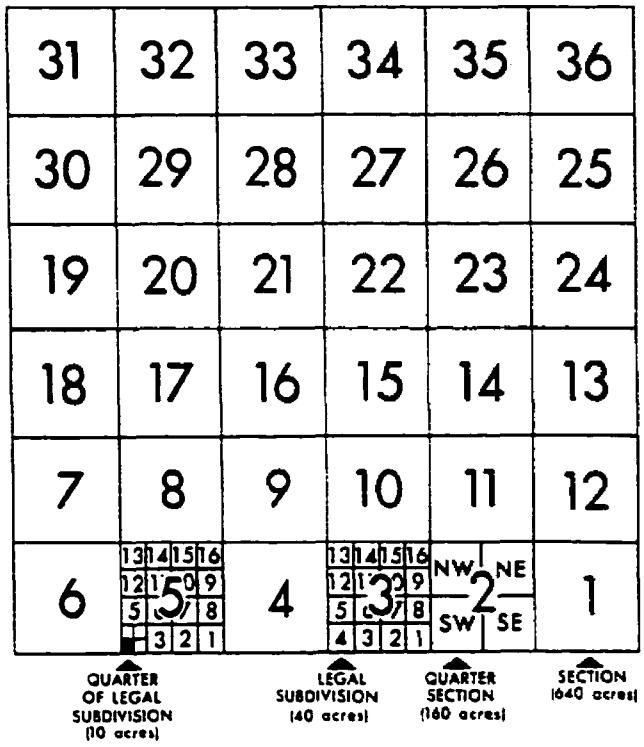


Figure 2.5 – Section Layout

Each section is then approximately 640 acres or 260 hectares in size. The legal description of a section can then be described with 4 numbers. The section number is always the first number in the 4 section sequence. 20-12-9 W4 is then section 20 of township 12, range 9 west of the 4th meridian.

Sections can further be subdivided 2 ways depending on the requirements. The simplest is to divide the section into quarter sections each labeled by their compass direction. A single section has quarter sections SE, SW, NE, NW. SW-20-12-9 W4 is then the south-west quarter section of section 20, township 12, range 9 west of the 4th. Each quarter section is approximately 160 acres or 65 hectares in size.

If the land is to be put to use where the quarter section is too large a subdivision, then the section can be divided into "Legal Subdivisions". A legal subdivision is 1/16 of a section, subdivided in the same manner as a township is divided into sections, starting from the south-east corner and winding to the north-east corner dividing the section into 16 parcels. Each legal subdivision is approximately 40 acres or 16 hectares. A legal subdivision can also be quartered giving the smallest subdivision of 10 acres or 4 hectares which can be expressed using the Dominion Land Survey System. Three numbers tell you that you are dealing with a township. Four numbers tell you that you are dealing with a section. Five numbers tell you that you are dealing with a legal subdivision. SW-20-12-9 W4 is then the south-west quarter section of section 20, township 12, range 9 west of the 4th. SW-15-20-12-9 W4 is then the south-west quarter of legal subdivision 15 of section 20, township 12, range 9 west of the 4th.

Attributes of Spatially Referenced Data

Understanding the Dominion Land Survey system is paramount to understanding the requirements of this specific project. Equally important is understanding the attributes and characteristics of spatially referenced data and Geographic Information Systems in general. Primary is an understanding of the two main methods for representing spatial data: Raster and Vector. A raster data structure is made of individual cells in a two dimensional row and column format (A.&C.E.T., 1997). Each cell is of the same size and that size determines the resolution of the data. It is akin to thinking of pixels on a computer screen. The second type of spatial data representation is vector. Vector structures make use of common geometrical figures such as points, lines and polygons to represent the spatial

characteristics of real world entities (A.&C.E.T., 1997). Primitive vector structures can be combined to make complex data representations. Using both vector or raster methods, attribute data can be layered (Burrough, 1996). That is, data that has common characteristics or a common theme can be grouped, stored and displayed separately. When the spatial data is analyzed certain layers can be included or excluded depending on the type of analysis that is occurring. Either method will result in a certain amount of interpolation due to the discrete nature of computer systems and the continuous nature of the real world (Burrough, 1996). This discreteness is represented by layering, a phenomenon which is contrary to reality.

Spatial Data References

Several texts deal with spatial data. "Fundamentals of Spatial Information Systems" by Robert Laurini and Derek Thompson (1996) is especially thorough at describing the more mathematical nature of spatial data. Several texts such as "Geographic Information Systems" by D.R. Fraser Taylor (1991) and "Principles of Geographical Information Systems for Land Resources Assessment" by P.A. Burrough (1996) excellently describe the GIS system which is the primary use of spatial data. "Spatial Analysis and Spatial Policy using Geographic Information Systems" by Les Worrall (1991) brings these two highly related topics together giving several excellent examples of the types of analysis that can be done from the spatially referenced data used in a GIS.

Requirements Specification Document for Client

The informal Requirements Specification Document (RSD) for this thesis was created as a result of a study done in conjunction with a major Oil and Gas company in Calgary. Most

of the requirements elicited deal with the method of using such a system, its inputs outputs and processes. The literature for spatial data systems was used to elicit requirements of a more technical nature as well as assist in probing areas of requirements that could be overlooked using strictly the users as the only source of data.

Autodesk MapGuide

The client for whom this study was done has already purchased a product for visual representation of their data. This is the Mapguide product from Autodesk (http://www.autodesk.com/products/mapguide). Mapguide permits the use of both Raster and Vector data files and provides an intelligent client system for viewing and manipulation. It comes complete with an engine for zooming and extracting data at any desired level. The following (figure 2.6) is a screen print from a Mapguide application showing data from a number of Alberta sections utilizing layered data in both raster and vector formats.

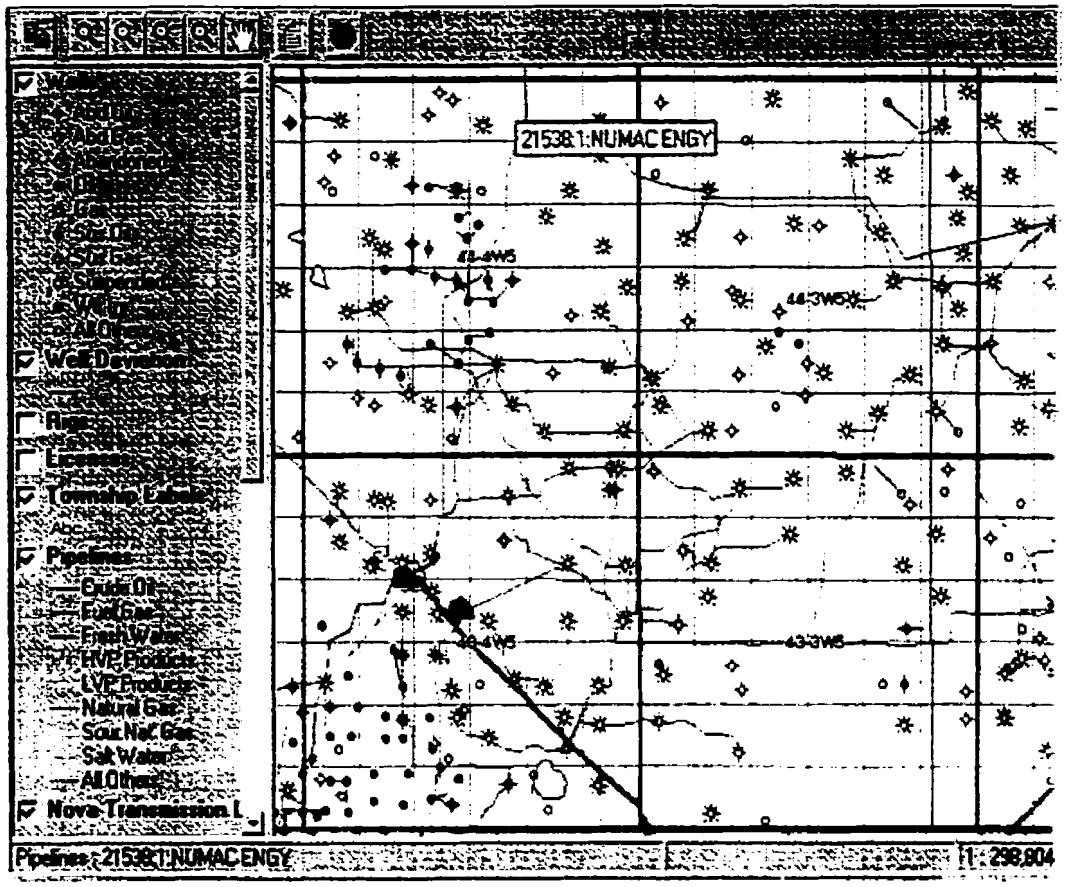


Figure 2.6 – Mapguide Sample

The Mapguide viewer is a stand alone application which would reside on each users desktop. That way each user can tailor their view of the data to their own preference. The data, however, is centrally stored. Each user, therefore, sees the same information. Data from vector file formats includes ESRI. SHP, ESRI. ARC/INFO, Intergraph, Mapinfo, Atlas and CSV. Raster file formats include GIF, TGA, CALS, PNG, BMP, JPEG and TIFF. These consist of all the common file formats used by most GIS packages available today.

2.3 Formal Methods

Formal Methods of requirements specification came into being as a result of the lack of precision and the presence of ambiguity in a narrative requirements specifications. A computer program should be correct. but it must be correct with respect to a specification. Computer programs are "formal" in the fact that they operate according to formal logic and discrete mathematics. In order to prove that a program is correct with respect to its abstract specification one must formalize its specification (Gibbins. 1990).

"A Formal Method is a set of rigorous engineering practices which is generally based on formal systems and which are applied to the development of engineering products such as software or hardware" (Gibbins. 1990). They apply logic and simple mathematics to programming (Jacky. 1997). Formal methods are by definition of a mathematical nature and work well in representing systems which have a high set orientation (Barden et al.. 1994). This is what led the author to believe that they would be applicable to the project of specifying a Land Management Information System. Formal methods. however. are not infallible. They cannot guarantee that software is perfect (Hall. 1990). They can in many cases augment traditional specification methods. Only rarely can a formal method be applied to all aspects of a system (Bowen. 1995).

Several formal specification languages exist. "A formal language is a language which has a precise set of semantics. as opposed to natural languages (English. French. German. Swedish) in which it is possible to construct ambiguous sentences. A formal language is needed to support a formal method." (Gibbons. 1990. p 1). CLEAR is a theory-oriented specification language. OBJ is an executable specification language. VDM and "Z" are

both non executable formal specification languages (Gibbons, 1990). The most common form of formal specifications is the "Z" language (Barden et al., 1994). It is based on the mathematical discipline of first-order logic and set theory (Diller, 1992). It is the "Z" language which will be used to express the formal specification within this thesis.

At the heart of the "Z" specification is a notation known as the "Schema". An example of a schema is shown in figure 2.7 (Stepney, 1995)

$[NAME, FILE]$

FileSys

$$fsys : NAME \pfun FILE$$
$$open : \mathbb{P}\, NAME$$

$$open \subseteq \mathrm{dom}\, fsys$$

Figure 2.7 – Schema Set Definitions

This particular schema is a file system specification which has a state consisting of a mapping from file names to file contents and the set of files open for reading. The declaration $[NAME, FILE]$ defines the set of all *NAME*'s and the set of all *FILE*'s as basic types in the specification. The identification at the top of the box "*FileSys*" is the name of the schema. The horizontal line divides the schema into 2 parts. The section above the line contains the declaratives. In the case of *FileSys* there are 2 declarations. The first declares the function *fsys* which when given a *NAME* returns a *FILE*. ie. *fsys(NAME) = FILE*. The second declaration *open:* $\mathbb{P}$ *NAME* defines the set *open* as the power set of *NAME*. That is, the set *open* can be any subset of the set of all elements of type *NAME*.

The area beneath the line consists of the predicate of the schema. In this case, the set *open* is said to be a subset of the domain of the function *fsys*.

This particular file system has a *Read* operation, as shown by the name of the following schema in figure 2.8, that takes a list of *NAME*'s and returns a corresponding list of *FILE*'s. For each file, if it is open, the operation gives the actual contents, but if it's closed, or does not exist, the operation gives an error (Stepney, 1995).

$| closed, doesNotExist : FILE$

Read

$$
\begin{array}{|l}
\hline
\Xi FileSys \\
n? : \mathrm{seq}\ NAME \\
f! : \mathrm{seq}\ FILE \\
\hline
\#f! = \#n? \\
\forall i : 1 .. \#n? \bullet \\
\quad (n? i \in open \Rightarrow f! i = fsys(n? i)) \\
\quad \wedge (n? i \in \mathrm{dom} fsys \setminus open \Rightarrow f! i = closed) \\
\quad \wedge (n? i \notin \mathrm{dom} fsys \Rightarrow f! i = doesNotExist) \\
\hline
\end{array}
$$

Figure 2.8 – Example Schema

Prior to the schema is a single declaration of 2 new sets, *closed* and *doesNotExist* of type *FILE*.

Operation schemata act on previously defined elements. In this case, the operation *Read* operates on the set *FileSys*. The symbol Ξ indicates that the set *FileSys* does not change as a result of the *Read* operation. If the operation did change the set then the symbol Δ would replace the Ξ indicating a change to the set was occurring as a result of the operation.

A question mark ("?") indicates input to an operation. In the case of the *Read* operation, *n*? indicates the input will be a sequence of *NAME*'s. An exclamation point ("!") indicates

output. *f*! indicates the output will consist of a sequence of *FILE*'s. That completes the declarations.

The first line of the predicate section of the *Read* operation states that the number of outputs will equal the number of inputs. The second statement is a conjunction. It can be read as follows: For every *i*, *i* ranging from 1 to the number of input *NAME*'s, if the *NAME* input, *n*?*i*, is an element of the set *open* then output *f*!*i*, the result of the function *fsys*(*n*?*i*). This will be of type *FILE* as defined in the *FileSys* schema. If *n*?*i* is not an element of *open* then check to see if *n*?*i* is an element of the set dom *fsys* \ *open* , that is, the difference between the set dom *fsys* and the set *open*. If that is the case, return *closed*. If neither of the first 2 predicates are satisfied then check that the input *n*?*i* is not an element of the set dom *fsys*, in which case, return *doesNotExist*.

Another common concept within "Z" is that of the tuple. Sets consist of a number of elements of the same type. Tuples, on the other hand, associate elements of any type in a fixed order (Jacky, 1997). For example, a date is represented by 3 related elements of different types, month, day and year.

DAY == 1..31; MONTH == 1..12; YEAR == ℤ

This says DAY can be from 1 to 31, MONTH from 1 to 12 and YEAR must be an Integer. If we order the tuple YEAR, MONTH, DAY then the tuple (2000,01,10) represents January 10, 2000. () Parentheses distinguish tuples from sets which employ braces {} (Jacky, 1997). Order is important in tuples. (2000,01,10) is not the same tuple as (2000,10,01) whereas {2000,01,10} and {2000,10,01} are the same set.

The tuple is first defined as a *Cartesian Cross Product*

DATE == YEAR X MONTH X DAY

It can be declared within an unnamed schema

```
start,end : DATE
-----------------
start = (2000,01,11)
end = (2000,06,02)
```

and then utilized in future schemata.

Updates to sets are indicated in "Z" by use of the override operator ⊕ (Spivey, 1992).

The symbol ↦ is defined within "Z" as the 'Maplet relation'. It is simply a graphical representation of an ordered pair. $x \mapsto y$ is therefore a graphic way of expressing (x,y) (Spivey, 1992).

For example, we could declare the set *NormalYear* = {*jan* ↦ 31,*feb* ↦ 28,*mar* ↦ 31, *apr* ↦ 30,*may* ↦ 31,*jun* ↦ 30,*jul* ↦ 31,*aug* ↦ 31,*sep* ↦ 30,*oct* ↦ 31,*nov* ↦ 30, *dec* ↦ 31}, that is, a set of relations between names of months and number of days. In order to define the set *LeapYear* all we would have to do is define *LeapYear* = *NormalYear* ⊕ {*feb* ↦ 29}. This would make *LeapYear* identical to *NormalYear* except for the fact that *LeapYear* would have *feb* with 29 days whereas *NormalYear* would have *feb* with 28 (Diller, 1992).

With regards to "Z" standards, from discussions with other formal methods practitioners (Stepney, 1998) , Spivey, (1992) is the current *de facto* standard. Spivey, being a major contributor to the development of the "Z" language, describes it in detail but is more of a

reference text. The literature describes "Z" quite well with several worked examples. (Bjorner. 1982). (Hayes. 1987). (Barden et al.. 1994). (Wordsworth. 1993) This thesis will document the process involved in getting from the standard informal narrative requirements specification to the integrated formal specification.

A good variety of "Z" literature can be found in academic circles and companies that specialize in formal methods. One such company. Logica UK. produces a product called "Formaliser" which is a "Z" composer and type checking tool.

Formaliser allows the user to compose "Z" schemata and include them in many types of documents. There is continuous type checking so that the user is notified of an error as it is typed and it can be corrected immediately. The following is an actual screen layout of a test "Z" schema included with the product for demonstration purposes.

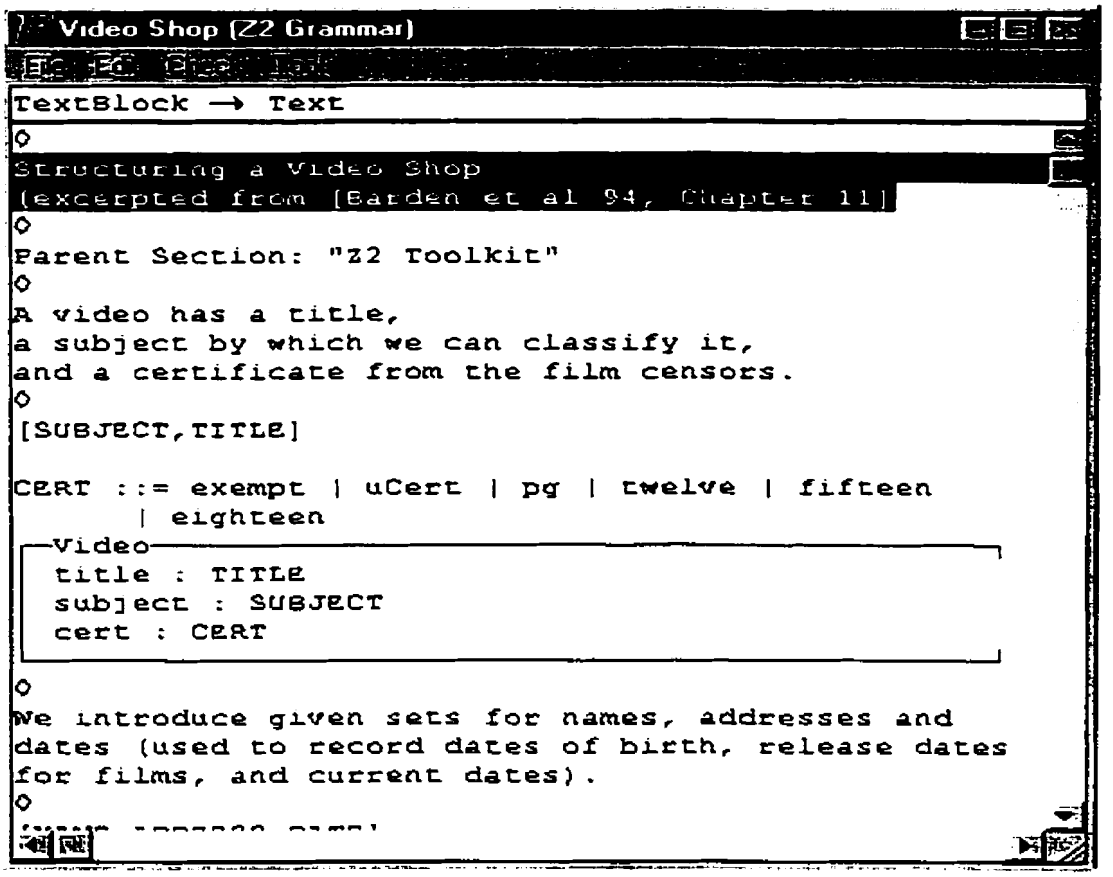


Figure 2.9 – Formaliser example

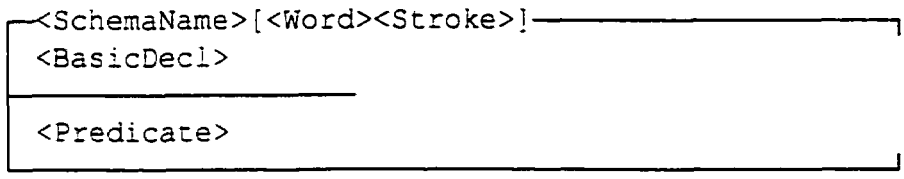
Formaliser runs under Microsoft Windows 3.1, 95 and NT. Output can be saved as text files suitable for processing by the LaTeX text formatting program or Microsoft Word or Corel WordPerfect. As can be seen from the above figure, Formaliser provides full support for schema notation and the full "Z" character set.

A Formaliser specification document is constructed through a mixture of structure editing and direct text entry (Logica, 1995). This document is held within Formaliser as a hierarchy of nodes. Nodes can represent text blocks, declarations, schemata, predicates or any other type of fully contained "Z" statement.

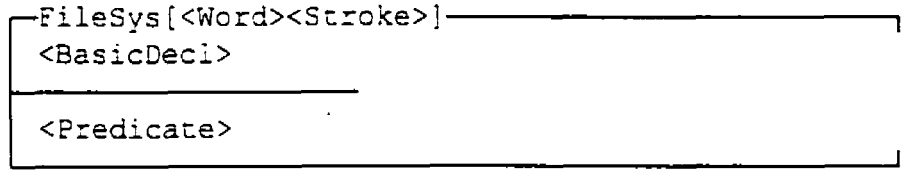
One would begin by inserting a node. It would appear in the Formaliser document as:

<Paragraph>

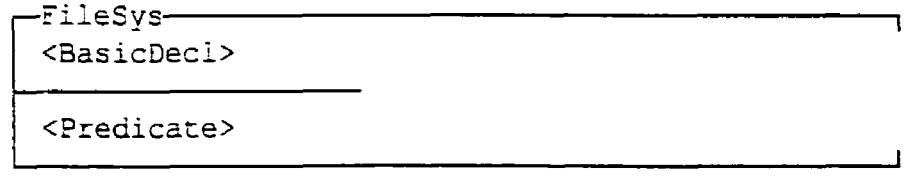
The next step is to instantiate that node. To do this, the type of the statement being inserted must be known. For example, if the node was instantiated as a "schema box", the node would change to



This gives the basic structure of a schema box. Notice that the one <Paragraph> node now consists of 5 different nodes. The SchemaName node requires no further refinement but clicking on it with the mouse allows you to type in the name of the schema, changing the box to



If the "Word" or "Stroke" nodes are not required, clicking on them and pressing the space bar results in



In order to create the declarations, the <BasicDecl> node must be instantiated further. Selecting the ColonDecl node type from the instantiation list changes <BasicDecl> to

<DeclName> : <Expression>. Clicking on <DeclName> and keying the declaration name yields fsys : <Expression>. Instantiating the node <Expression> as an InFixGenericExpression yields fsys : <Expression1> <InGen><Stroke> <Expression1>. Clicking on the first <Expression1> and typing NAME. clicking on the second <Expression1> and typing FILE. clicking on <Stroke> and hitting the space bar yields fsys : NAME <InGen> FILE.

Formaliser contains a complete symbol palette for all the "Z" symbols. Clicking on <InGen> and clicking on "Symbol Palette" brings up the following window:

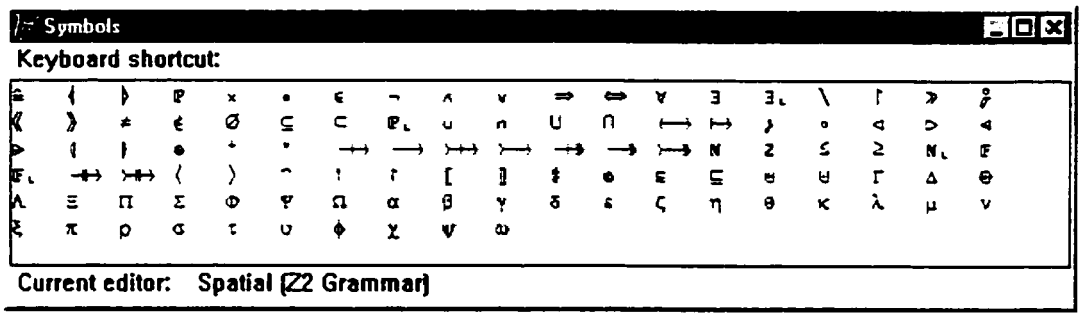


Figure 2.10 – Formaliser symbols

Doubleclick on the symbol $\rightarrowtail$ and it will be inserted in the highlighted area of the document resulting in the schema now looking like:

```
┌─FileSys──────────────────────────────────┐
│ fsys : NAME ⤖ FILE                       │
├──────────────                            │
│ <Predicate>                              │
└──────────────────────────────────────────┘
```

Highlighting the entire declaration node and pressing the "Append Node" command yields;

```
FileSys
fsys : NAME ⇸ FILE
<BasicDecl>
---------------
<Predicate>
```

The process would then be repeated for the second declarative node. The steps of node insertion, instantiation is repeated until the schema is complete. At that point a new node would be inserted after the entire schema and another schema can be build in the same manner. Formal "Z" specifications of any size can be created this way and type checking is done "on the fly" meaning that the specification must be valid at all times. Declarations must therefore precede operations.

2.4 Summary

This chapter has explored the relevant literature with regards to the three significant areas of this document, namely: Requirements Engineering, Spatial Data and Formal Methods. Each of these three disciplines is mature in its own right but their integration appears to be unexplored as of yet.

There are many different methods of Requirements Engineering but their goal is the same, that being to produce a specification for a system. The particular method being used for this thesis is that of a modified "Structured Analysis". This is due not to any particular fit but to the fact that it is the method used by the client for whom this particular Requirements Specification Document is being produced. The primary tools for "Structured Analysis" are the Dataflow Diagram and the Entity Relationship Diagram, both of which are explored.

This thesis explores the Formal Specification methods for a system employing Spatially Referenced Data. This too is an area requiring special attention. This chapter presented a synopsis of the Dominion Land Survey system which is used in the province of Alberta, Canada, where this study is taking place. Understanding of the DLS is paramount to understanding the requirements. It is the framework within which the requirements are elicited and specified.

Basic information concerning the nature of spatial data is also presented. The difference between Raster and Vector presentation is explored as is the nature of the "layering" of data.

The Mapguide product is introduced as it is the preselected engine which will be used in the development of this product.

The final section of this chapter deals with Formal Methods. The questions as to what are Formal Methods and why are they used are answered. The "Z" language is the formal specification language which will be used for the formal specification of this Requirements Specification Document so it is introduced as to its notation and syntax. The product "Formaliser", which will be used for the production of the "Z" specification, is also described.

Chapter 3

3.0 Integrating Informal and Formal Requirements Methods; A Practical Approach for Systems Employing Spatially Referenced Data

Appendix A of this document contains a complete informal narrative Requirements Specification Document as presented to and accepted by a client. This chapter extends that Document by expressing the requirements formally using the "Z" Specification Language. The intention of this exercise is to see if integrating Formal Specifications into the Informal Narrative Specification results in a reduction of ambiguity and an increase in precision. The "Z" specification is followed by a formal proof of a schema and a table showing in which schema each of the informal requirements is specified.

Introduction

The method used to develop the "Z" specifications will be based on the method described in "Z in Practice" by Rosalind Barden. Susan Stepney and David Cooper (1994). The resulting specification will consist of the following;

1. The given sets and global constants for the specification. together with an informal description of their significance.
2. A schema that describes the abstract state. If the state is complex different parts should be determined in separate schemata. then combined using the schema calculus.
3. A schema that describes the initial state of the system.
4. Schemata that describe the abstract operations on the state.

5. Information to assist the reader of the specification, for example a cross reference list of schema names.

The method used consists of examining and analyzing each of the requirements from Appendix A. in order, and determining if each requirement represents a functional or non functional requirement. Functional requirements result in a schema specifying the operation required to fulfill the function. If the schema is operating on data not yet represented in an abstract state. an abstract state schema is also created representing the data. If the schema representing the record has already been specified it is determined if all data required to fulfill the function is specified. If it is. then the operation is specified. If it isn't. the new required data is added to the record schema as well as the operation schema. Any new functions or relations are added to the ProtoDatabase schema and Database schema. Any new files are added to the InitDatabase schema and any new responses are added to the list of valid responses. To satisfy the audit requirements. any new change descriptions are added to the CHANGEDESC set. Finally text is added to describe the schema.

Non functional requirements that either do not require or cannot be specified are described in informal text.

This "Z" specification describes a Land Management Information System. It was produced using "Formaliser" available from Logica UK Ltd.

3.1 Given Sets and Global Constants

The system controls a set of projects each having a Project Name and Start Date.

```
[PROJNAME, DATE]
```

Sets of quartersection designations and Legal Land Descriptions must be associated to an individual project.

```
[QSNAME, LLD]
```

The following basic data types can be associated with People and Legal Land Descriptions.

```
[PERSON, ADDRESS, CONTACTDATA]
```

In order to satisfy requirement CR8 – Provide Audit Trail, there must be an audit log which contains user id's, current date and time and a description of changes made.

```
[USERID, DATETIME]

CHANGEDESC ::= addproj | deleteproj | modifyproj | addqtrs
             | deleteqtrs | addperson | deleteperson
             | addlld | deletelld | addllddesc | modllddesc
             | dellddesc | addcompensation | addcontdates
             | appraise | addform | modform | delform
             | assignform | updatecontactdata
             | addencroachment
```

Encroachments are requests made by third parties to cross or utilize and existing Right-of-Way.

[ENCROACHMENT]

Forms are maintained by the Legal Group and are associated with Projects and LLD's.

[FORM, FORMDETAIL, FORMPACKAGE]

Null Datatypes must be specified.

```
| NULLDATE : DATE
| NULLFORM : FORM
| NULLFORMS : ℙ FORM
| NULLPACKAGE : FORMPACKAGE
| NULLPACKAGES : ℙ FORMPACKAGE
| NULLLLDS : ℙ LLD
| NULLQTRS : ℙ QSNAME
| NULLPROJECT : ProjectRecord
| NULLPERSON : PersonRecord
| NULLPEOPLE : ℙ PersonRecord
```

All possible responses must be specified.

```
Responses ::= success | notactive | notentered |
alreadyactive | alreadyentered | cannotassociate |
notassociated | knownperson | unknownperson | formassigned |
alreadydescribed | notdescribed | formexists |
formdoesnotexist | compensationexists | cannotassign
Response == ℙ Responses
```

3.2 Abstract States

The following declarations define the datatypes within the system that will be used as records.

The LogRecord contains the userid. datetime stamp and a description of every modification to the database.

```
LogRecord
loguser : USERID
logchangedt : DATETIME
logchange : CHANGEDESC
```

LLD's have several descriptive attributes which will be combined under the data type LandDesc. These are land size. purchase price. last sold date and Canada Land Index.

```
LandDesc
LandSize : ℕ
LandPrice : ℕ
LandSoldDate : DATE
LandCLI : ℕ
```

```
LLDRecord
lld : LLD
lldDesc : LandDesc
```

Quartersections can contain more than one LLD.

```
QuartersectionRecord
desc : QSNAME
llds : ℙ LLD
```

A Project record consists of a Project name, a start date and a list of Quartersections associated with that project.

```
ProjectRecord
name : PROJNAME
startdate : DATE
quartersections : ℙ QSNAME
```

People are Landowners, Occupants or anyone required to be known to the system. They possess an address and contact information as well as the set of forms and form packages that person can receive.

```
PersonRecord
name : PERSON
proj : PROJNAME
lld : LLD
where : ADDRESS
contact : CONTACTDATA
forms : ℙ FORM
packages : ℙ FORMPACKAGE
```

Forms are contained within their own set to be maintained by the Legal Group.

```
FormRecord
formname : FORM
formdetail : FORMDETAIL
formpackage : FORMPACKAGE
```

Compensations are based on the Project and the LLD. They determine what is owed to whom and when.

```
CompensationRecord
CompProj : PROJNAME
CompLLD : LLD
CompPerson : PERSON
CompStartDate : DATE
CompDueDate : DATE
CompStopDate : DATE
CompAmt : N
```

Encroachments are kept together keyed by Project ID and LLD.

```
EncroachmentRecord
EncProj : PROJNAME
EncLLD : LLD
Encrcachment : ENCROACHMENT
EncStartDate : DATE
EncStopDate : DATE
```

Tasks are a combination of Project ID and LLD. They are used to determine the required tasks of the operations department.

```
Task
TaskProj : PROJNAME
TaskLLD : LLD
```

Cheques contain a persons name. address. cheque date and amount.

```
Cheque
Cheqname : PERSON
Cheqaddr : ADDRESS
Cheqdate : DATE
Cheqamt : N
```

The 'ProtoDatabase' is a schema which contains the records and files associated with the project. It contains the abstract state, the declaration that ActiveProjects are projects for which the startdate has a valid value as well as constraints that project and person records have unique names. Functions are contained within the 'Database' schema.

```
ProtoDatabase
LogFile : ℙ LogRecord
ProjectFile : ℙ ProjectRecord
PersonFile : ℙ PersonRecord
LLDFile : ℙ LLDRecord
QuartersectionFile : ℙ QuartersectionRecord
FormFile : ℙ FormRecord
CompensationFile : ℙ CompensationRecord
EncroachmentFile : ℙ EncroachmentRecord
ActiveProjects : ℙ ProjectRecord
---
ActiveProjects
 = { x : ProjectRecord
        x ∈ ProjectFile ∧ x.startdate ≠ NULLDATE }
∀ x,y : ProjectRecord • x.name = y.name ⇒ x = y
∀ x,y : PersonRecord • x.name = y.name ∧ x.lld = y.lld
        ⇒ x = y
```

The following function prototypes are declared.

```
SymbolDefinition: relation ( AlreadyActive _ )
SymbolDefinition: relation ( AlreadyEntered _ )
SymbolDefinition: relation ( LLDAlreadyDescribed _ )
SymbolDefinition: relation ( _ KnownPerson _ )
SymbolDefinition: relation ( ExistingForm _ )
SymbolDefinition: relation ( AssignedForm _ )
SymbolDefinition: relation ( AssignedPackage _ )
SymbolDefinition: relation ( _ AlreadyAssociated _ )
SymbolDefinition: relation ( _ AlreadyDescribed _ )
```

The 'Database' Schema contains the 'ProtoDatabase' as well as all the function implementations.

```
┌─Database──────────────────────────────────────────────────────┐
│ ProtoDatabase
│ GetDBProj : PROJNAME ⟶ ProjectRecord
│ GetDBPersonbyProj : PROJNAME ⟶ PersonRecord
│ AlreadyActive _ : ℙ PROJNAME
│ AlreadyEntered _ : ℙ PROJNAME
│ LLDAlreadyDescribed _ : ℙ LLD
│ _ KnownPerson _ : ℙ PERSON
│ ExistingForm _ : ℙ FORM
│ AssignedForm _ : ℙ FORM
│ AssignedPackage _ : ℙ FORM
│ _ AlreadyAssociated _ : QSNAME ⟶ PROJNAME
│ _ AlreadyDescribed _ : QSNAME ⟶ LLD
├──────────────────────────────────────────────────────────────
│ ∀ n : PROJNAME •
│   ((∃ p : ProjectRecord • p.name = n) ⇒
│     (∀ p : ProjectRecord | p.name = n •
│        GetDBProj n = p))
│   ∧ (¬ (∃ p : ProjectRecord • p.name = n) ⇒
│       GetDBProj n = NULLPROJECT)
│ ∀ n : PROJNAME •
│   ((∃ p : PersonRecord • p.proj = n) ⇒
│     (∀ p : PersonRecord | p.proj = n •
│        GetDBPersonbyProj n = p))
│   ∧ (¬ (∃ p : PersonRecord • p.proj = n) ⇒
│       GetDBPersonbyProj n = NULLPERSON)
│ ∀ n : PROJNAME •
│   AlreadyActive n
│   ⇔ (∃ p : ProjectRecord | p ∈ ActiveProjects •
│        p.name = n)
│ ∀ n : PROJNAME •
│   AlreadyEntered n
│   ⇔ (∃ p : ProjectRecord | p ∈ ProjectFile •
│        p.name = n)
│ ∀ l : LLD •
│   LLDAlreadyDescribed l
│   ⇔ (∃ lr : LLDRecord | lr ∈ LLDFile • lr.lld = l)
│ ∀ n : PERSON; l : LLD •
```

```
  l KnownPerson n
  ⇔ (∃ p : PersonRecord | p ∈ PersonFile •
       p.name = n ∧ p.lld = l)
∀ f : FORM •
  ExistingForm f
  ⇔ (∃ fr : FormRecord | fr ∈ FormFile •
       fr.formname = f)
∀ f : FORM •
  AssignedForm f
  ⇔ (∃ fr : FormRecord; pr : PersonRecord |
          fr ∈ FormFile ∧ pr ∈ PersonFile •
       fr.formname = f ∧ f ∈ pr.forms)
∀ f : FORM •
  AssignedPackage f
  ⇔ (∃ fr : FormRecord; pr : PersonRecord |
          fr ∈ FormFile ∧ pr ∈ PersonFile •
       fr.formname = f
       ∧ fr.formpackage ∈ pr.packages)
∀ n : PROJNAME; q : QSNAME •
  q AlreadyAssociated n
  ⇔ (∃ p : ProjectRecord | p ∈ ProjectFile •
       p.name = n
       ∧ (∃ qs : QSNAME | qs ∈ p.quartersections •
            qs = q))
∀ q : QSNAME; l : LLD •
  q AlreadyDescribed l
  ⇔ (∃ qs : QuartersectionRecord |
          qs ∈ QuartersectionFile •
       qs.desc = q
       ∧ (∃ lld : LLD | lld ∈ qs.llds • lld = l))
```

3.3 Initial States

Initially the Database is empty.

InitDatabase

Database

ProjectFile = Ø
LogFile = Ø
PersonFile = Ø
LLDFile = Ø
QuartersectionFile = Ø
FormFile = Ø
CompensationFile = Ø
EncroachmentFile = Ø

3.4 Abstract Operations

Projects can be added if they are not already entered. In order to satisfy CR8 – Provide Audit Trail. all changes to all sets require an addition to an audit log file.

Schema 1.

```
AddProject
ΔDatabase
p? : PROJNAME
d? : DATE
dt? : DATETIME
user? : USERID
r! : Response
─────────────────────────────────────────────
"Add new Project Record if not already entered"
¬ AlreadyEntered p? ⇒
 (∃ new : ProjectRecord •
    new.name = p? ∧ new.startdate = d?
    ∧ new.quartersections = NULLQTRS
    ∧ ProjectFile' = ProjectFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addproj
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if new project already exists"
AlreadyEntered p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {alreadyentered}

"Remaining Files are unchanged"
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Projects can be modified (have their start date changed) if they are entered and the date to change to is not NULL.

Schema 2.

```
ChangeProject
ΔDatabase
p? : PROJNAME
notnulldate? : DATE
dt? : DATETIME
user? : USERID
r! : Response
-------------------------------------------------
"Change existing project if entered"
AlreadyEntered p? ⇒
 (∃ old,new : ProjectRecord |
        old.name = p? ∧ old ∈ ProjectFile •
    new.name = old.name ∧ new.startdate = notnulldate?
    ∧ new.quartersections = old.quartersections
    ∧ ProjectFile' = ProjectFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = modifyproj
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if project does not exist"
¬ AlreadyEntered p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notentered}

"Remaining Files are unchanged"
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Projects can be deleted if they exist and are not active.

Schema 3.

```
DeleteProject
ΔDatabase
p? : PROJNAME
proj : ProjectRecord
dt? : DATETIME
user? : USERID
r! : Response
---
"Delete project if already entered"
AlreadyEntered p? ∧ ¬ AlreadyActive p? ⇒
 proj = GetDBProj p?
 ∧ ProjectFile' = ProjectFile \ {proj}
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = deleteproj
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if project does not exist"
¬ AlreadyEntered p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notentered}

"Produce an error message if project to delete is active"
AlreadyActive p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {active}

"Remaining Files are unchanged"
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Abstract Operations Schemata 1. 2 and 3 specify Construction Requirements 1. Projects may now be added. modified and deleted.

Quartersections must now be associated with individual active projects if they are not already associated.

Schema 4.

```
AddQuarterSection
ΔDatabase
p? : PROJNAME
q? : QSNAME
dt? : DATETIME
user? : USERID
r! : Response
---
"Associate quartersection with project if project is"
"active and quartersection is not already associated"
"Create a new quartersection record if one does not exist"
AlreadyActive p? ∧ ¬ q? AlreadyAssociated p? ⇒
 (∃ old,new : ProjectRecord |
        old.name = p? ∧ old ∈ ProjectFile •
    new.name = old.name
    ∧ new.startdate = old.startdate
    ∧ new.quartersections
       = old.quartersections ∪ {q?}
    ∧ ProjectFile' = ProjectFile \ {old} ∪ {new})
 ∧ (∃ old,new : QuartersectionRecord |
         old.desc = q? ∧ old ∈ QuartersectionFile •
      new.desc = q? ∧ new.llds = NULLLLDS
      ∧ QuartersectionFile'
         = QuartersectionFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addqtrs
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if project is not active"
¬ AlreadyActive p? ∧ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notactive}
```

```
"Produce error message if project is active but"
"quartersection is already associated"
AlreadyActive p? ∧ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {alreadyassociated}
"Produce error message if trying to associate an already"
"associated quartersection to an inactive project"
¬ AlreadyActive p? ∧ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notactive,alreadyassociated}

"Remaining Files are unchanged"
PersonFile' = PersonFile
LLDFile' = LLDFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

QuarterSections can be deleted from specific projects if the project is active and the quartersection is already associated.

Schema 5.

```
DeleteQuarterSection
ΔDatabase
p? : PROJNAME
q? : QSNAME
dt? : DATETIME
user? : USERID
r! : Response
```

```
"Delete quartersection if project is active and"
"quartersection is associated with project"
AlreadyActive p? ∧ q? AlreadyAssociated p? ⇒
 (∃ old,new : ProjectRecord |
        old.name = p? ∧ old ∈ ProjectFile ••
    new.name = old.name
    ∧ new.startdate = old.startdate
    ∧ new.quartersections
```

```
        = old.quartersections \ {q?}
    ∧ ProjectFile' = ProjectFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = deleteqtrs
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if project is not active"
¬ AlreadyActive p? ∧ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notactive}

"Produce error message if project is active but"
"quartersection is not associated"
AlreadyActive p? ∧ ¬ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notassociated}

"Produce error message if trying to delete a non"
"associated quartersection from a non active project"
¬ AlreadyActive p? ∧ ¬ q? AlreadyAssociated p? ⇒
 ProjectFile' = ProjectFile ∧ LogFile' = LogFile
 ∧ r! = {notactive,notassociated}

"Remaining Files are unchanged"
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Abstract Operation Schemata 4 & 5 specify Construction Requirements 2. Quarter Sections can now be added and deleted from active projects.

The list of QuarterSections must now be sent to the Alberta Registry Office so that all Legal Land Descriptions within those QuarterSections can be determined and added to the project.

Given a specific project. produce a list of all associated QuarterSections.

Schema 6.

```
GetQuarterSections
EDatabase
p? : PROJNAME
ToLRIS! : ℙ QSNAME
r! : Response
-------------------------------------------
"Produce a list of quartersections for a project if"
"project is active"
AlreadyActive p? ⇒
 ToLRIS! = (GetDBProj p?).quartersections
 ∧ r! = {success}

"Produce an error message if the project is not active"
¬ AlreadyActive p? ⇒ r! = {notactive}
```

The set that is retrieved from the government registry will contain all the Legal land Descriptions and Landowner information for each QuarterSection requested. This data must now be associated with appropriate QuarterSections.

Associated with each QuarterSection sent to the registry is returned the Legal Land Descriptions. Landowners address and contact data. Occupants address and contact data. land sizes. purchase prices. last sold dates and Canada Land Indexes.

Add the people, addresses and contact data to the People file. People may or may not be associated with any particular LLD. There must exist the ability to delete people as well in case of changes in ownership etc.

Schema 7.

```
AddPerson
ΔDatabase
p? : PERSON
pr? : PROJNAME
l? : LLD
a? : ADDRESS
c? : CONTACTDATA
dt? : DATETIME
user? : USERID
r! : Response
```

"Add person if not already known and project is active"
¬ l? KnownPerson p? ∧ AlreadyActive pr? ⇒
 (∃ new : PersonRecord •
    new.name = p? ∧ new.proj = pr? ∧ new.lld = l?
    ∧ new.where = a? ∧ new.contact = c?
    ∧ new.forms = NULLFORMS
    ∧ new.packages = NULLPACKAGES
    ∧ PersonFile′ = PersonFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addperson
      ∧ LogFile′ = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if project is not active"
¬ AlreadyActive pr? ∧ l? KnownPerson p? ⇒
 PersonFile′ = PersonFile ∧ LogFile′ = LogFile
 ∧ r! = {notactive}

"Produce error message if person is already known"
l? KnownPerson p? ∧ AlreadyActive pr? ⇒
 PersonFile′ = PersonFile ∧ LogFile′ = LogFile
 ∧ r! = {knownperson}

"Produce error message if person is already known"

```
"and project is not active"
l? KnownPerson p? ∧ ¬ AlreadyActive pr? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {notactive,knownperson}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Schema 8.

```
DeletePerson
ΔDatabase
p? : PERSON
pr? : PROJNAME
l? : LLD
dt? : DATETIME
user? : USERID
r! : Response
---
"Delete person if already known"
l? KnownPerson p? ⇒
 (∃ old : PersonRecord |
        old.name = p? ∧ old.proj = pr?
        ∧ old.lld = l? ∧ old ∈ PersonFile •
    PersonFile' = PersonFile \ {old})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = deleteperson
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if person is unknown"
¬ l? KnownPerson p? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {unknownperson}

"Remaining Files are unchanged"
```

```
ProjectFile' = ProjectFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Add the Legal land Descriptions to the QuarterSection file associated with the appropriate QuarterSection. This can be done either as a result of the file returned from the government registry or manually in the cases where the data did not exist within the registry.

Schema 9.

```
AddLLDtoQuartersection
ΔDatabase
q? : QSNAME
l? : LLD
dt? : DATETIME
user? : USERID
r! : Response
---
"Add LLD to Quartersection if not already described"
¬ q? AlreadyDescribed l? ⇒
 (∃ old,new : QuartersectionRecord |
        old.desc = q? ∧ old ∈ QuartersectionFile •
    new.desc = old.desc
    ∧ new.llds = old.llds ∪ {l?}
    ∧ QuartersectionFile'
       = QuartersectionFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addlld
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error if already described"
q? AlreadyDescribed l? ⇒
 QuartersectionFile' = QuartersectionFile
 ∧ LogFile' = LogFile ∧ r! = {alreadydescribed}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
```

```
LLDFile' = LLDFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Schema 10.

```
DeleteLLDfromQuartersection
ΔDatabase
q? : QSNAME
l? : LLD
dt? : DATETIME
user? : USERID
r! : Response
---
"Delete LLD from quartersection if already described"
q? AlreadyDescribed l? ⇒
 (∃ old,new : QuartersectionRecord |
        old.desc = q? ∧ old ∈ QuartersectionFile •
    new.desc = old.desc
    ∧ new.llds = old.llds \ {l?}
    ∧ QuartersectionFile'
       = QuartersectionFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = deletelld
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if quartersection is not already"
"described"
¬ q? AlreadyDescribed l? ⇒
 QuartersectionFile' = QuartersectionFile
 ∧ LogFile' = LogFile ∧ r! = {notdescribed}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

At this point projects are being created and made active. QuarterSections are being associated with those projects and Legal Land Descriptions and landowner/Occupant data are being associated with those QuarterSections.

The Legal Land Description is now described. Full functionality of add, modify and delete is required.

Schema 11.

```
AddLandDesc
ΔDatabase
l? : LLD
ld? : LandDesc
dt? : DATETIME
user? : USERID
r! : Response
---
"Describe LLD if not already described"
- LLDAlreadyDescribed l? ⇒
 (∃ new : LLDRecord •
    new.lld = l? ∧ new.lldDesc = ld?
    ∧ LLDFile' = LLDFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addlldesc
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if LLD is already described"
LLDAlreadyDescribed l? ⇒
 LLDFile' = LLDFile ∧ LogFile' = LogFile
 ∧ r! = {alreadydescribed}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Schema 12.

```
ModLandDesc
ΔDatabase
l? : LLD
ld? : LandDesc
dt? : DATETIME
user? : USERID
r! : Response
---
"Modify LLD if already described"
LLDAlreadyDescribed l? ⇒
 (∃ old,new : LLDRecord |
        old.lld = l? ∧ old ∈ LLDFile •
    new.lld = old.lld ∧ new.lldDesc = ld?
    ∧ LLDFile' = LLDFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = modlldesc
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if LLD is not already described"
¬ LLDAlreadyDescribed l? ⇒
 LLDFile' = LLDFile ∧ LogFile' = LogFile
 ∧ r! = {notdescribed}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Schema 13.

```
DelLandDesc
ΔDatabase
l? : LLD
dt? : DATETIME
user? : USERID
r! : Response
---
"Delete LLD description if already described"
LLDAlreadyDescribed l? ⇒
 (∃ old : LLDRecord | old.lld = l? ∧ old ∈ LLDFile •
    LLDFile' = LLDFile \ {old})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = delLlddesc
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if LLD is not described"
¬ LLDAlreadyDescribed l? ⇒
 LLDFile' = LLDFile ∧ LogFile' = LogFile
 ∧ r! = {notdescribed}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

The Construction Group has the ability to enter Contract dates into the system either by Project Name or Project LLD key. First a Compensation record must be created for each Project. LLD. Person combination.

Schema 14.

```
AddCompensationRecord
ΔDatabase
p? : PROJNAME
l? : LLD
person? : PERSON
dt? : DATETIME
user? : USERID
r! : Response
---
"Add a new compensation record keyed by project, lld and"
"person if it does not exist"
(∃ new : CompensationRecord •
   new.CompProj = p? ∧ new.CompLLD = l?
   ∧ new.CompPerson = person?
   ∧ new ∉ CompensationFile) ⇒
 (∃ new : CompensationRecord •
    new.CompProj = p? ∧ new.CompLLD = l?
    ∧ new.CompPerson = person?
    ∧ new.CompStartDate = NULLDATE
    ∧ new.CompDueDate = NULLDATE
    ∧ new.CompStopDate = NULLDATE ∧ new.CompAmt = 0
    ∧ CompensationFile' = CompensationFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addcompensation
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the compensation record"
"already exists"
(∃ new : CompensationRecord •
   new.CompProj = p? ∧ new.CompLLD = l?
   ∧ new.CompPerson = person?
   ∧ new ∈ CompensationFile) ⇒
 CompensationFile' = CompensationFile
 ∧ LogFile' = LogFile ∧ r! = {compensationexists}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
```

Schema 15.

```
EnterContractDatesbyLLD
ΔDatabase
p? : PROJNAME
l? : LLD
startdate? : DATE
duedate? : DATE
stopdate? : DATE
dt? : DATETIME
user? : USERID
r! : Response
```

```
"Add contract dates if the project is active"
AlreadyActive p? ⇒
 (∃ old,new : CompensationRecord |
        old.CompProj = p? ∧ old.CompLLD = l?
        ∧ old ∈ CompensationFile •
    new.CompProj = p? ∧ new.CompLLD = l?
    ∧ new.CompPerson = old.CompPerson
    ∧ new.CompStartDate = startdate?
    ∧ new.CompDueDate = duedate?
    ∧ new.CompStopDate = stopdate?
    ∧ new.CompAmt = old.CompAmt
    ∧ CompensationFile'
       = CompensationFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addcontdates
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the project is not active"
¬ AlreadyActive p? ⇒
 CompensationFile' = CompensationFile
 ∧ LogFile' = LogFile ∧ r! = {notactive}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
```

Schema 16.

```
EnterContractDatesEnMass
ΔDatabase
p? : PROJNAME
startdate? : DATE
duedate? : DATE
stopdate? : DATE
dt? : DATETIME
user? : USERID
r! : Response
---
"Update all compensation records for this project if the"
"project is active"
AlreadyActive p? ⇒
 CompensationFile'
  = CompensationFile
    \ { old : CompensationRecord |
          old.CompProj = p? ∧ old ∈ CompensationFile }
    ∪ { new : CompensationRecord |
          ∀ old : CompensationRecord |
            old.CompProj = p? ∧ old ∈ CompensationFile •
            new.CompProj = p?
            ∧ new.CompLLD = old.CompLLD
            ∧ new.CompPerson = old.CompPerson
            ∧ new.CompStartDate = startdate?
            ∧ new.CompDueDate = duedate?
            ∧ new.CompStopDate = stopdate?
            ∧ new.CompAmt = old.CompAmt }
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addcontdates
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the project is not active"
- AlreadyActive p? ⇒
 CompensationFile' = CompensationFile
 ∧ LogFile' = LogFile ∧ r! = {notactive}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
```

At this point in the project, all QuarterSections are known and all available data connected with them is known and entered into the system.

This completes the formal specification for the first series of tasks for the Construction Group. Their next task is to inform the Appraisal Group of the new project so they can determine fair compensation for each landowner/Occupant for the project.

Appraisals are for each LLD for each project. A particular LLD may have several contracts associated with it depending on how many projects run through that particular piece of property. Appraisals are therefore keyed by Project and LLD.

Schema 17.

```
AppraisebyLLD
ΔDatabase
p? : PROJNAME
l? : LLD
appraisal? : ℕ
dt? : DATETIME
user? : USERID
r! : Response
─────────────────────────────────────────
"Update the compensation record with a new appraisal by"
"LLD if the project is active"
AlreadyActive p? ⇒
 (∃ old,new : CompensationRecord |
        old.CompProj = p? ∧ old.CompLLD = l?
        ∧ old ∈ CompensationFile •
    new.CompProj = p? ∧ new.CompLLD = l?
    ∧ new.CompPerson = old.CompPerson
    ∧ new.CompStartDate = old.CompStartDate
    ∧ new.CompDueDate = old.CompDueDate
    ∧ new.CompStopDate = old.CompStopDate
    ∧ new.CompAmt = appraisal?
    ∧ CompensationFile'
       = CompensationFile \ {old} ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = appraise
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the project is not active"
- AlreadyActive p? ⇒
 CompensationFile' = CompensationFile
 ∧ LogFile' = LogFile ∧ r! = {notactive}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
```

Schema 18.

```
AppraiseEnMass
ΔDatabase
p? : PROJNAME
appraisal? : ℕ
dt? : DATETIME
user? : USERID
r! : Response
---
"Modify all compensation records for a project with a"
"new appraisal if the project is active"
AlreadyActive p? ⇒
 CompensationFile'
  = CompensationFile
    \ { old : CompensationRecord |
            old.CompProj = p? ∧ old ∈ CompensationFile }
    ∪ { new : CompensationRecord |
            ∀ old : CompensationRecord |
                 old.CompProj = p?
                 ∧ old ∈ CompensationFile •
              new.CompProj = p?
              ∧ new.CompLLD = old.CompLLD
              ∧ new.CompPerson = old.CompPerson
              ∧ new.CompStartDate
                 = old.CompStartDate
              ∧ new.CompDueDate = old.CompDueDate
              ∧ new.CompStopDate = old.CompStopDate
              ∧ new.CompAmt = appraisal? }
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = appraise
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the project is not active"
¬ AlreadyActive p? ⇒
 CompensationFile' = CompensationFile
 ∧ LogFile' = LogFile ∧ r! = {notactive}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
```

This completes the specification for the appraisal process. LLD's can now be appraised by project either individually or en masse.

The Appraisal Group then informs the Construction Group that the appraisals are complete.

The Construction Group informs the Legal Group that a new project requires legal forms. It is one of the tasks of the Legal Group to maintain the set of available forms.

Forms consist of a form name and details as stored within the Forms database. Forms may be added, changed or deleted.

Schema 19.

```
AddForm
ΔDatabase
f? : FORM
fd? : FORMDETAIL
fp? : FORMPACKAGE
dt? : DATETIME
user? : USERID
r! : Response
─────────────────────────────────────
"Add Form if it does not exist"
¬ ExistingForm f? ⇒
 (∃ new : FormRecord •
    new.formname = f? ∧ new.formdetail = fd?
    ∧ new.formpackage = fp?
    ∧ FormFile' = FormFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addform
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if form already exists"
ExistingForm f? ⇒
 FormFile' = FormFile ∧ LogFile' = LogFile
 ∧ r! = {formexists}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Schema 20.

```
MaintainForm
ΔDatabase
f? : FORM
fd? : FORMDETAIL
fp? : FORMPACKAGE
dt? : DATETIME
user? : USERID
r! : Response
---
"Maintain form if it exists"
ExistingForm f? ⇒
 (∃ old,new : FormRecord |
        old.formname = f? ∧ old ∈ FormFile •
    new.formname = f? ∧ new.formdetail = fd?
    ∧ new.formpackage = fp?
    ∧ FormFile' = FormFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = modform
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce error message if form to maintain does not exist"
¬ ExistingForm f? ⇒
 FormFile' = FormFile ∧ LogFile' = LogFile
 ∧ r! = {formdoesnotexist}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

In order for a form to be deleted, it cannot be currently in use within any project.

Schema 21.

```
DeleteForm
ΔDatabase
f? : FORM
dt? : DATETIME
user? : USERID
r! : Response
─────────────────────────────────────────
"Delete form if it exists and it or its package is not"
"assigned anywhere"
ExistingForm f? ∧ ¬ AssignedForm f?
∧ ¬ AssignedPackage f? ⇒
 (∃ old : FormRecord |
        old.formname = f? ∧ old ∈ FormFile •
    FormFile' = FormFile \ {old})
 ∧ (∃ l : LogRecord ••
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = delform
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the form to delete does not"
"exist"
¬ ExistingForm f? ⇒
 FormFile' = FormFile ∧ LogFile' = LogFile
 ∧ r! = {formdoesnotexist}

"Produce an error message if the form to delete exists"
"but is currently being used"
ExistingForm f? ∧ (AssignedForm f? ∨ AssignedPackage) f?⇒
 FormFile' = FormFile ∧ LogFile' = LogFile
 ∧ r! = {formassigned}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Forms that are maintained by the Legal Group can be assigned to a specific Project/LLD combination. These can be changed at any time by the Legal Group.

Schema 22.

```
AssignFormbyLLD
ΔDatabase
p? : PROJNAME
lld? : LLD
f? : FORM
fp? : FORMPACKAGE
dt? : DATETIME
user? : USERID
r! : Response
```

```
"Assign the form to an LLD if the form and project both"
"exist"
AlreadyActive p? ∧ ExistingForm f? ⇒
 PersonFile' = PersonFile
   \ { old : PersonRecord |
       old.proj = p? ∧ old.lld = lld? ∧ old ∈ PersonFile }
   ∪ { new : PersonRecord |
            ∀ old : PersonRecord |
              old.proj = p? ∧ old.lld = lld?
              ∧ old ∈ PersonFile •
              new.name = old.name ∧ new.proj = p?
              ∧ new.lld = lld?
              ∧ new.where = old.where
              ∧ new.contact = old.contact
              ∧ new.forms = old.forms ∪ {f?}
              ∧ new.packages = old.packages ∪ {fp?} }
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = assignform
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the project is not active"
¬ AlreadyActive p? ∧ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {notactive}

"Produce an error message if the form does not exist"
```

```
AlreadyActive p? ∧ ¬ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {formdoesnotexist}

"Produce an error message if the project is not active"
"and the form does not exist"
AlreadyActive p? ∧ ¬ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {notactive,formdoesnotexist}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

Forms can also be assigned to persons not associated with a project. These can be pre created forms or custom forms.

Schema 23.

```
AssignFormbyPerson
ΔDatabase
pr? : PERSON
l? : LLD
f? : FORM
fp? : FORMPACKAGE
dt? : DATETIME
user? : USERID
r! : Response
---
"Assign a form to a person if the person and form both"
"exist"
l? KnownPerson pr? ∧ ExistingForm f? ⇒
 PersonFile' = PersonFile \
 { old : PersonRecord |old.name = pr? ∧ old ∈ PersonFile }
    ∪ { new : PersonRecord |
        ∀ old : PersonRecord |
          old.name = pr? ∧ old ∈ PersonFile •
```

```
            new.name = pr? ∧ new.proj = old.proj
            ∧ new.lld = old.lld ∧ new.where = old.where
            ∧ new.contact = old.contact
            ∧ new.forms = old.forms ∪ {f?}
            ∧ new.packages = old.packages ∪ {fp?} }
  ∧ (∃ l : LogRecord •
       l.loguser = user? ∧ l.logchangedt = dt?
       ∧ l.logchange = assignform
       ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if the person does not exist"
¬ l? KnownPerson pr? ∧ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {unknownperson}

"Produce an error message if the form does not exist"
l? KnownPerson pr? ∧ ¬ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {formdoesnotexist}

"Produce an error message if the person is unknown"
"and the form does not exist"
¬ l? KnownPerson pr? ∧ ¬ ExistingForm f? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {unknownperson,formdoesnotexist}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

At this point the Legal Group has designated all documentation that has to be completed for this project. The Construction Group is then informed of task completion and they must then carry out the obligations.

Construction Group obligations at this point consist of printing all required documentation. Contact data is retrieved from the set of People and is inserted into the forms as required.

Schema 24.

```
┌─DetermineDocumentation──────────────────────────────────┐
│ ΞDatabase                                               │
│ p? : PROJNAME                                           │
│ ToPrinter! : ℙ FORM                                     │
│ r! : Response                                           │
├─────────────────────────────────────────────────────────┤
│ "Determine documentation if the project is active"      │
│ AlreadyActive p? ⇒                                      │
│  ToPrinter! = (GetDBPersonbyProj p?).forms              │
│  ∧ r! = {success}                                       │
│                                                         │
│ "Produce an error message if the project is not active" │
│ ¬ AlreadyActive p? ⇒ r! = {notactive}                   │
└─────────────────────────────────────────────────────────┘
```

All required documentation for the project is printed. This is then given to the Land Agents for distribution.

The Land Agents require access to all contact information in order to deliver the documentation.

Schema 25.

```
┌─GetContactData──────────────────────────────────────────┐
│ ΞDatabase                                               │
│ p? : PERSON                                             │
│ l? : LLD                                                │
│ ContactData! : ℙ CONTACTDATA                            │
│ r! : Response                                           │
├─────────────────────────────────────────────────────────┤
│ "Get contact data for land agent if person is known"    │
│ l? KnownPerson p? ⇒                                     │
│  ContactData!                                           │
│   = { contact : CONTACTDATA |                           │
│           ∀ pr : PersonRecord |                         │
│                 pr.name = p? ∧ pr ∈ PersonFile •
```

```
            contact = pr.contact } ∧ r! = {success}

"Produce error message if person is unknown"
¬ l? KnownPerson p? ⇒ r! = {unknownperson}
```

The Land Agent has a requirement to update the Contact Data with the results of the contact made with the recipient of the documents. Included in the contact data is an acknowledgment of the delivery of required documentation.

Schema 26.

```
UpdateContactData
ΔDatabase
pr? : PROJNAME
l? : LLD
p? : PERSON
cd? : CONTACTDATA
dt? : DATETIME
user? : USERID
r! : Response
─────────────────────────────────────
"Update the contact data from the land agent if this is"
"a known person"
l? KnownPerson p? ⇒
 (∃ old,new : PersonRecord |
        old.name = p? ∧ old.proj = pr?
        ∧ old.lld = l? ∧ old ∈ PersonFile •
    new.name = p? ∧ new.proj = pr? ∧ new.lld = l?
    ∧ new.where = old.where ∧ new.contact = cd?
    ∧ new.forms = old.forms
    ∧ new.packages = old.packages
    ∧ PersonFile' = PersonFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = updatecontactdata
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}

"Produce an error message if this is an unknown person"
¬ l? KnownPerson p? ⇒
 PersonFile' = PersonFile ∧ LogFile' = LogFile
 ∧ r! = {unknownperson}

"Remaining Files are unchanged"
ProjectFile' = ProjectFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
```

```
CompensationFile' = CompensationFile
EncroachmentFile' = EncroachmentFile
```

At this point control of the project is ready to be handed over to the Operations Group. One of the tasks of the Operations Group is to pay amounts due to people to whom the company has a contractual obligation. In order to accomplish this task they must be informed of all obligation 30 days before they are due.

Schema 27.

```
DetermineTasks
ΞDatabase
ThirtyDaysFromNow : DATE
ToDo! : P Task
r! : Response
---
"Determine tasks for next thirty days"
ToDo!
 = { Task   ∀ cr : CompensationRecord |
                  cr.CompDueDate
                   = ThirtyDaysFromNow •
              TaskProj = cr.CompProj
              ∧ TaskLLD = cr.CompLLD }
r! = {success}
```

On the day that the task is due. write a cheque to each eligible recipient. After the cheque has been written. update the due date for that compensation to one year from today.

Schema 28.

```
WriteCheque
ΔDatabase
today? : DATE
ThirtyDaysFromToday : DATE
OneYearFromToday : DATE
ToPrinter! : ℙ Cheque
r! : Response
---
ToPrinter! = { ChequestoPrint : Cheque |
   ∀ cr : CompensationRecord | cr.CompDueDate = today? •
          ChequestoPrint.Cheqamt = cr.CompAmt
          ∧ ChequestoPrint.Cheqname
             = cr.CompPerson
          ∧ ChequestoPrint.Cheqdate
             = cr.CompDueDate
          ∧ (∃ pr : PersonRecord |
                pr.name = cr.CompPerson
                ∧ pr ∈ PersonFile •
             ChequestoPrint.Cheqaddr = pr.where) }
CompensationFile' = CompensationFile
   \ { old : CompensationRecord |
       old.CompDueDate = today? ∧ old ∈ CompensationFile }
   ∪ { new : CompensationRecord |
          ∀ old : CompensationRecord |
               old.CompDueDate = today?
               ∧ old ∈ CompensationFile •
            new.CompProj = old.CompProj
            ∧ new.CompLLD = old.CompLLD
            ∧ new.CompPerson = old.CompPerson
            ∧ new.CompStartDate
               = old.CompStartDate
            ∧ new.CompDueDate
               = OneYearFromToday
            ∧ new.CompStopDate = old.CompStopDate
            ∧ new.CompAmt = old.CompAmt }
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
EncroachmentFile' = EncroachmentFile
r! = {success}
```

Record encroachments that are received for any specific project.

Schema 29.

```
AddEncroachment
ΔDatabase
e? : ENCROACHMENT
lld? : LLD
p? : PROJNAME
startdate? : DATE
stopdate? : DATE
dt? : DATETIME
user? : USERID
r! : Response
```

"Add encroachment to an active project"

```
AlreadyActive p? ⇒
 (∃ new : EncroachmentRecord •
    new.EncProj = p? ∧ new.EncLLD = lld?
    ∧ new.Encroachment = e?
    ∧ new.EncStartDate = startdate?
    ∧ new.EncStopDate = stopdate?
    ∧ EncroachmentFile' = EncroachmentFile ∪ {new})
 ∧ (∃ l : LogRecord •
      l.loguser = user? ∧ l.logchangedt = dt?
      ∧ l.logchange = addencroachment
      ∧ LogFile' = LogFile ∪ {l}) ∧ r! = {success}
```

"Produce error message if project is not active"

```
¬ AlreadyActive p? ⇒
 EncroachmentFile' = EncroachmentFile
 ∧ LogFile' = LogFile ∧ r! = {notactive}
```

"Remaining Files are unchanged"

```
ProjectFile' = ProjectFile
PersonFile' = PersonFile
LLDFile' = LLDFile
QuartersectionFile' = QuartersectionFile
FormFile' = FormFile
CompensationFile' = CompensationFile
```

Encroachments are registered with the Provincial Government by sending them to the Provincial Registry.

Schema 30.

```
RegisterEncroachments
EDatabase
p? : PROJNAME
ToGov! : ℙ ENCROACHMENT
r! : Response
-------------------------------------------------
"Register encroachments with the government for active"
"projects"
AlreadyActive p? ⇒
 ToGov! = { n : ENCROACHMENT |
          ∀ er : EncroachmentRecord |
               er.EncProj = p? •
            n = er.Encroachment } ∧ r! = {success}

"Produce error message if project is not active"
¬ AlreadyActive p? ⇒ r! = {notactive}
```

3.5 Proving the specification

Proving a specification is not intended to prove that the specification properly represents the informal requirements. That is impossible to do. It is intended to verify that the Formal Specification is correct within itself (Barden et al., 1994). That is to say, it is "internally consistent". The value of determining the fact that the Formal Specification is internally consistent is that it is the Formal Specification that will be used to design the application, not the informal narrative specification. A Formal Specification that is internally consistent is one that can be constructed. Proving a specification is a very laborious and expensive undertaking (Barden et al., 1994) and a complete proof of this Formal Specification is

beyond the scope of this thesis although there is evidence that conducting an extensive proof is more effective at finding errors than extensive testing (King, 2000). One proof will be included to demonstrate the technique. The technique is as demonstrated in Bergmann et al (1980). Several rules will be used.

The first will be that of "Assumption". Assumptions are true statements given as part of the theorem being proven. They are stated at the beginning of each level of scope with the proof following.

A "Tautology" is simply a statement that is always true. It differs from an assumption in that it will usually consist of a disjunction of opposites. Something will be true or its opposite will be true.

Conjunction Elimination "$\wedge$-Elimination" removes terms from a conjunctive statement. Conjunction Elimination, Simplification, Equality and Rewrite steps are all intended to simplify the proof by removing unused information, changing the order of information without altering its meaning or renaming information to names used later on in the proof. The statements associated with these rules is logically equivalent to the statements from which they were derived.

An "Implication" is a relationship between two propositions in which the second is the logical consequence of the first. If it is true that "If A then B" and A is true, then B can be derived..

The technique of "Constructive Dilemma" is used to complete the proof. Constructive Dilemma tells us that if we have 2 separate assumptions in a disjunction and both assumptions logically imply a third statement that that statement can be taken as true.

The Schema that will be proved is Schema 4 – AddQuarterSection. It will show that when the AddQuarterSection operation is executed adding QuarterSection QS to Project PN, the result will be the relation QS AlreadyAssociated PN.

Proof

Prove that adding Quartersection QS to Project PN results in a state of Quartersection QS being AlreadyAssociated with Project PN.

PN = Project Name

QS = LLD of quartersection being added

Now = Current datetime stamp

Userid = User id of user performing operation

Response = Response from abstract operation

Theorem (AddQuartersection PN QS Now Userid Response ⇒

QS AlreadyAssociated PN)

1.	AddQuartersection PN QS Now Userid Response	Assumption
2.	QS AlreadyAssociated PN ∨ ¬ QS AlreadyAssociated PN	Tautology
3.	QS AlreadyAssociated PN	Assumption (AddQuartersection's precondition fails)
4.	(∃ p : ProjectRecord | p ∈ ProjectFile • p.name = PN ∧ (∃ qs : QSNAME | qs ∈ p.quartersections • qs = QS))	Definition of AlreadyAssociated
5.	¬ QS AlreadyAssociated PN	Assumption (AddQuartersection's precondition passes)

6. (∃ old.new : ProjectRecord | old.name = p? ∧ old ∈ ProjectFile • new.name = old.name ∧ new.startdate = old.startdate ∧ new.quartersections = old.quartersections ∪ {q?} ∧ ProjectFile′ = ProjectFile \ {old} ∪ {new}) ∧ (∃ old.new : QuartersectionRecord | old.desc = q? ∧ old ∈ QuartersectionFile • new.desc = q? ∧ new.llds = NULLLLDS ∧ QuartersectionFile′ = QuartersectionFile ∪ {new}) ∧ (∃ l : LogRecord • l.loguser = user? ∧ l.logchangedt = dt? ∧ l.logchange = addqtrs ∧ LogFile′ = LogFile ∪ {l}) ∧ r! = {success}

1.5. Definition of AddQuartersection

7. (∃ old.new : ProjectRecord | old.name = p? ∧ old ∈ ProjectFile • new.name = old.name ∧ new.startdate = old.startdate ∧ new.quartersections = old.quartersections ∪ {q?} ∧ ProjectFile′ = ProjectFile \ {old} ∪ {new})

6. ∧-Elimination

8. (∃ old.new : ProjectRecord | old.name = p? ∧ old ∈ ProjectFile • new.name = old.name ∧ new.quartersections = old.quartersections ∪ {q?} ∧ ProjectFile′ = ProjectFile \ {old} ∪ {new})

7. ∧-Elimination

9. (∃ old.new : ProjectRecord | old ∈ ProjectFile • new.name = p? ∧ new.quartersections = old.quartersections ∪ {q?} ∧ ProjectFile′ = ProjectFile \ {old} ∪ {new})

8. Equality

10. (∃ old.new : ProjectRecord | old ∈ ProjectFile • new.name = p? ∧ new.quartersections = old.quartersections ∪ {q?} ∧ new ∈ ProjectFile′)

9. Simplification

11. (∃ old.new : ProjectRecord | old ∈ ProjectFile • new.name = p? ∧ q? ∈ new.quartersections ∧ new ∈ ProjectFile′)

10. Simplification

12.	(∃ new : ProjectRecord | new.name = p? ∧ q? ∈ new.quartersections ∧ new ∈ ProjectFile′)	11, Simplification. removal of "old"
13.	(∃ new : ProjectRecord | new ∈ ProjectFile′ • new.name = p? ∧ q? ∈ new.quartersections	12. Reorder
14.	(∃ new : ProjectRecord | new ∈ ProjectFile • new.name = p? ∧ q? ∈ new.quartersections)	13. Rewrite "ProjectFile′"/ "ProjectFile"
15.	(∃ new : ProjectRecord | new ∈ ProjectFile • new.name = PN ∧ QS ∈ new.quartersections)	14. Rewrite "p?"/PN. "q?"/QS
16.	(∃ p : ProjectRecord | p ∈ ProjectFile • p.name = PN ∧ QS ∈ p.quartersections	15. Rewrite "new"/"p"
17.	QS ∈ p.quartersections	Assumption
18.	(∃ qs : QSNAME | qs ∈ p.quartersections • qs = QS))	17. ∃ Introduction
19.	QS ∈ p.quartersections ⇒ (∃ qs : QSNAME | qs ∈ p.quartersections • qs = QS))	17-18. Implication Introduction
20.	(∃ p : ProjectRecord | p ∈ ProjectFile • p.name = PN ∧ (∃ qs : QSNAME | qs ∈ p.quartersections • qs = QS))	16.19. Implication
21.	(∃ p : ProjectRecord | p ∈ ProjectFile • p.name = n ∧ (∃ qs : QSNAME | qs ∈ p.quartersections • qs = q))	2, 3-4, 5-20 Constructive Dilemma
22.	QS AlreadyAssociated PN	21, Definition of AlreadyAssociated

23. AddQuartersection PN QS Now Userid Response ⇒ QS AlreadyAssociated PN — 1-22, Implication

QED

3.6 Requirements Disposition

Each of the narrative requirements from the Requirements Specification Document in Appendix A is listed in the matrix that follows (Table 3.1). Next to the requirement is the disposition of that requirement. That can be either: N/A if the requirement could not be or was not intended to be expressed formally, or the Schema designation from this chapter.

Table 3.1 – Requirements Disposition through Formal Methods

Requirements from Narrative RSD	Disposition
General Requirement 1 – Ease of Use	**N/A**
General Requirement 2 – Y2K Compliant	**N/A**
General Requirement 3 - Secure	**N/A**
Construction Requirement 1 – Add Project	**Schema 1,2,3**
Construction Requirement 2 – Assign Quartersections	**Schema 4,5**
Construction Requirement 3 – Assign Subdivisions	**Schema 9**
Construction Requirement 4 – Receive landowners	**Schema 6**
Construction Requirement 5 – Parse LRIS List	**N/A**
Construction Requirement 6 – Record Land Data	**Schema 7**
Construction Requirement 7 – Manually enter land data	**Schema 11**

Construction Requirement 8 – Provide Audit Trail	**All**
Construction Requirement 9 – Notify Appraisals	**N/A**
Construction Requirement 10 – Notify Construction Group	**N/A**
Construction Requirement 11 – Select Forms	**Schema 24**
Construction Requirement 12 – Package Forms	**Schema 22,23**
Construction Requirement 13 – Designate Recipients	**Schema 23**
Construction Requirement 14 – Enter Data en-mass	**Schema 16**
Construction Requirement 15 – Notify Operations Group	**N/A**
Appraisal Requirement 1 – Notify Appraisals	**N/A**
Appraisal Requirement 2 – Access to Data	**N/A**
Appraisal Requirement 3 – Assign Compensation	**Schema 17,18**
Appraisal Requirement 4 – Mass Update	**Schema 18**
Appraisal Requirement 5 – Notify Requesting Group	**N/A**
Operations Requirement 1 – Retrieve Obligations	**Schema 27**
Operations Requirement 2 – Get Obligation Data	**N/A**
Operations Requirement 3 – Update Landowner Data	**Schema 7,8**
Operations Requirement 4 – Notify Appraisal Group	**N/A**
Operations Requirement 5 – Print Documentation	**Schema 24**
Operations Requirement 6 – Record Encroachment Requests	**Schema 29**
Operations Requirement 7 – Create Custom Forms	**Schema 19,20,21**
Operations Requirement 8 – Register Encroachments	**Schema 30**

Land Agent Requirement 1 – Access Contact Data	Schema 25
Land Agent Requirement 2 – Access from Remote Locations	N/A
Land Agent Requirement 3 – Record New Contact Data	Schema 26
Land Agent Requirement 4 – Acknowledge Documentation	Schema 26
Legal Requirement 1 – Notification of Project	N/A
Legal Requirement 2 – Access to Government Data	N/A
Legal Requirement 3 – Designate Documentation	Schema 22,23
Legal Requirement 4 – Direct Documentation	Schema 22
Legal Requirement 5 – Third Party Documentation	Schema 23
Legal Requirement 6 – Maintain Document List	Schema 19
Legal Requirement 7 – Package Documentation	Schema 19
Legal Requirement 8 – Create Templates	Schema 19,20,21
Legal Requirement 9 – Determine Data Source for Templates	Schema 19
External Interface requirement 1 – Windows NT 4.0	N/A
External Interface requirement 2 – Seamless Access to LMIS	N/A
External Interface requirement 3 – Access to Autodesk Mapguide	N/A
External Interface requirement 4 – Seemless Access to StrataWeb	N/A
External Interface requirement 5 – StrataWeb Data	N/A
Performance requirement 1 – Response Time	N/A

3.7 Summary

This chapter formally specified, using the "Z" formal specification language, the informal requirements for a Land Management Information System as found in Appendix A. In the next chapter, this formal specification will be compared to the informal specification to determine if areas of ambiguity or lack of precision were discovered in the informal specification as a result of the formal specification.

Chapter 4

4.0 Evaluation of Specification Method

This chapter presents an evaluation of the Requirements Specification Document created through the integration of Informal and Formal Specification Methods. In particular, it seeks to discover errors or ambiguity in the informal requirements document found in Appendix A as a result of specifying the informal requirements formally using the "Z" formal specification language.

Did the creation of a formal "Z" Specification using Formaliser result in errors or omissions being detected in the informal narrative Requirements Specification Document? This can only be determined by a re-examination of the Requirements Specification Document in Appendix A and comparing it with the Formal Specification from Chapter 3. Not all requirements from Appendix A could, or should be, specified formally. A formal specification is not intended to replace a narrative RSD but to compliment it where best suited. (Barden, 1994).

Requirements CR1 – Add Project

Requirement CR1 states – "When the Construction Group is notified that a new project is being considered, they must be able to designate a name and number for that project. All relevant project information can be obtained using that unique identification."

This is the only reference to creating a set of projects. The ambiguity and lack of precision in this statement is not readily apparent. The "Z" specification details all the operations

that would be required to properly carry out this function including Add, Modify and Delete and all error conditions. This is specified in Schema 1, AddProject, Schema 2, ChangeProject and Schema 3, DeleteProject. It is precise and unambiguous. A better rewording of Requirement CR-1 would be:

CR1 – *When the Construction Group is notified that a new project is being considered, they must be able to add that new project identification and description to the set of projects if it has not been already entered. This function must also allow modification of existing project names and deletion of an existing project prior to it being assigned a valid start date. A project will be considered active once a valid start date has been assigned to it.*

Requirement CR3 – Assign Subdivisions

Requirement CR3 deals with the providing of a list of quartersections to the Alberta Government in order to retrieve a list of landowner data. No method of retrieving that list is given. The requirement could be satisfied by simply giving the user a free format screen on which a list of quartersections could be typed, which would be unsatisfactory. The "Z" specification, Schema 6, GetQuarterSections, shows how the Project ID must be entered in order to retrieve the data . A more precise stating of that requirement would be:

CR3 – *The system must include a function through which the user may select or enter a project ID and have returned a list of all associated quartersections in a format that can be passed to the Alberta Government Land Registry Information System for processing.*

Requirement CR7 – Manually enter land data

Requirement CR7 deals with the manual entry of landowner/occupant data as it becomes available. Can this data be modified if it currently exists? Schema 12, ModLandDesc of the "Z" specification says yes, the narrative RSD is ambiguous. CR7 is better reworded as follows:

CR7 – *Not all required information is returned via the provincial LRIS. A function must exist which allows the user to create, modify or delete required quartersection attribute data.*

Requirement CR8 – Provide Audit Trail

CR8 states that an audit trail is required, but for which functions? This is clearly shown in the "Z" specification as all schemata that modify data create an appropriate Log Record.

Requirement CR14 – Enter Data en-mass

CR14 deals with entering contract data per landowner. It is not clear that the key to this data is Project-LLD and that Add, Modify and Delete functions are required. This is clearly stated in the "Z" specification as Schema 16, EnterContractDatesEnMass. CR14 is better reworded as follows:

CR14 – *The Construction Group must be able to enter, update and delete contract start dates, contract terms and end dates for each landowner, either individually or en-mass. Individual data must be referenced by Project Name and LLD whereas en-mass updates must be able to be done by Project Name only.*

Requirement AR4 – Mass Update

AR4 does not mention that the key for appraisal amounts is Project-LLD. This is clearly defined in Schema 18. AppraiseEnMass. A more precise requirement would state:

AR4 - *Entering an amount per hectare and selecting the affected LLD's from a list of all LLD's associated with the selected project shall update appraisal amounts for all parcels.*

Requirement OR1 – Retrieve Obligations

Requirement OR1 does not state what data is required for it's report. This is only determined in Schema 27. DetermineTasks. OR1 is better reworded as follows:

OR1 – *It is one of the tasks of the Operations Group to satisfy contractual obligations. They must therefore have a list of all Project names and LLD's on a daily basis for which contractual obligations are due within the next 30 calendar days.*

Requirement OR4 – Notify Appraisal Group

OR4 states that the Operations Group must have the ability to notify the Appraisal group when a new appraisal is required. It is not clear as to what data is to be passed. They must be notified with Project ID and LLD. This is clear from the data returned by Schema 27. DetermineTasks. OR4 should be reworded as follows:

OR4 – *The Operations Group must have the ability to notify the Appraisal Group at contract renewal time of any Project Name, LLD combinations for which a revised appraisal is required. They must also have the ability to be notified by the Appraisal Group when the appraisal is complete and updated.*

4.1 Analysis

The informal narrative Requirement Specification in Appendix A has 50 documented requirements. As a result of the Formal Specification of these requirements using "Z", 8, or 16%, of the requirements were modified.

Requirement CR1 is both incomplete and ambiguous. It does not specify all required functionality to complete the requirement.

Requirement CR3 is incomplete in that it does not specify the data required to complete the task.

Requirement CR7 sufferes from ambiguity in that as it is originally informally stated it leaves room for interpretation.

Requirement CR8 dealt with the requirement for an audit trail. It is incomplete in that it does not state under what circumstances an audit trail is required.

Requirement CR14 is both incomplete and ambiguous in that it does not state what functionality is required and leaves open to interpretation what information is required to complete it.

Requirement AR4 is incomplete in that it does not state under what circumstances changes may be made to the database.

Requirement OR1 is incomplete in that it does not state what information is required.

Requirement OR4 is also incomplete in that it does not state what information is required to complete the requirement.

It was believed by the client that the narrative Requirements Specification Document was complete, accurate and unambiguous. It was the intention of that client to construct a

system using the informal Requirement Specification as the description of the requirements for this new system. Its Formal Specification using "Z" has shown that there would have been questions raised or assumptions made during the project's development. Delays would have resulted for several reasons. In cases of incompleteness, the person developing the design from the informal Requirement Specification would have had to returned to the Requirements Engineer in order to get clarification as to what was actually required. In cases of ambiguity, the designer would either have returned to the Requirements Engineer for clarification or made assumptions as to what was required. If assumptions were made in the design phase which did not satisfy the client, extensive and expensive development could take place which would be unacceptable to the client. Expensive rework could then result. Delays would have resulted in later phases of the project in order to resolve these ambiguous situations.

4.2 Summary

Chapter 4 examined the informal Requirements Specification Document found in Appendix A to try and discover ambiguity, incompleteness and lack of precision. It accomplished this through a comparison of the requirements in Appendix A with their Formal Specification from Chapter 3. Eight examples of ambiguity and incompleteness were discovered and documented.

Chapter 5

5.0 Conclusions

Did an increase in precision and a decrease in ambiguity take place within the Requirements Specification Document with the introduction of Formal Methods?

In order to draw conclusions, the aim and objectives as stated in Chapter 1 must be examined as to whether they were achieved or not. From Chapter 1, the aim of this document was "to show that when a formal specification is incorporated into the Requirements Specification Document for an LMIS system, the specification that results will be of greater precision". Chapter 4 determined that the Requirements Specification Document for an LMIS system found in Appendix A did suffer from a lack of precision and that the Requirements Engineer could decrease that lack of precision through an examination of a Formal Specification of those requirements.

Objective 1 was to understand the current state of the art. This was divided into three distinct disciplines, namely: Requirements Engineering, Spatial Data systems and Formal Methods. This was accomplished in Chapter 2. The subject of Requirements Engineering is extensive and the current state of the art was taken within the context of creating an informal narrative style Requirements Specification Document. An understanding of Spatial Data systems was also taken within the context of creating an Requirements Specification Document for a Land Management Information System within the Oil and Gas industry in the province of Alberta, Canada. Understanding Formal Methods was

achieved by developing the ability to create "Z" specification using the Formaliser application purchased from Logica UK Ltd.

Objective 2 was to develop an informal narrative RSD for a Land Management Information System. This objective was clearly satisfied as Appendix A of this document as this was a reduced Requirements Specification Document as accepted for the client for whom this project was developed.

Objective 3 was to develop a method of integrating informal and formal requirements methods for systems employing spatially referenced data. This was the main objective of this thesis, developing a formal specification for the system as defined through the informal Requirements Specification Document from Appendix A. Chapter 3 is the Formal Specification of that system. The narrative text interspersed with the "Z" code integrates the two forms of specification. It should also be noted that by using the "Z" specification product "Formaliser", not only was the benefit of specifying the requirements formally shown but the benefit of specifically using "Z" through the rigor of this particular specification product.

Objective 4 is a critical examination of the combined results of Chapter 3 and Appendix A. This is accomplished as Chapter 4. Chapter 4 shows that the creation of the formal specification and the application of the precise specifications against the informal Requirements Specification Document resulted in changes to 8 of 50 requirements.

The Requirements Specification Document as specified in Appendix A was as accepted by the client for whom this project was being developed. The fact that this was accepted showed that no ambiguity or lack of precision was initially evident. The client felt that

Appendix A properly defined the requirements for the system they desired. The creation of the Formal "Z" Specification identified ambiguity and lack or precision not initially evident. Since the cost of correcting errors increases with the stage in which they're discovered (Gause & Weinberg. 1989), this thesis shows that by utilizing Formal Methods. errors and omissions can be discovered earlier in the system development process and therefore will result in a more accurate. cheaper and faster system development.

There is a tradeoff in that extensive time is spent in the creation of the Formal Specification. It could extend the Requirements phase of a project by many months. For this particular project. the time spent creating the formal specification was approximately equal to the time spent in creation of the informal requirements document.

There is also evidence that constructing a complete proof of a specification can find additional errors (King. 2000). Only a single proof was included in this thesis whereas a complete proof of the entire specification would increase the cost and time spent in the Requirements phase substantially. It is beyond the scope of this thesis to determine if there is a threshold beyond which the cost to find errors is greater than finding them in subsequent development or testing phases.

In conclusion. this thesis began in Chapter 1 with a clearly stated aim and 4 related objectives. In each of the subsequent chapters. that aim and each of the objectives were successfully satisfied and can be summarized as follows:

"The integration of Informal and Formal requirements methods for systems employing spatially referenced data can result in a Requirements Specification Document that is less

ambiguous and more precise than a Requirements Specification Document constructed using informal methods alone".

5.1 Future Work

The fact that the cost and time to correct errors increases with the amount of time they are undetected, combined with the fact that the integration of Formal Methods into the Requirements Specification phase of this project resulted in a decrease in ambiguity, an increase in precision and a more complete specification, lends credence to the ascertain that Formal Methods are a valuable tool in the Requirements Elicitation phase. Companies that are involved with determining a project's requirements would be well advised to investigate utilizing Formal Methods in an effort to improve this phase and shorten the overall length of a project. This ignores any effort that may be involved in the learning and application of Formal Methods. Future work would be required to determine if the effort required to integrate Informal and Formal Methods offsets the gains in the increases in precision and reduction of ambiguity.

This thesis deals with a specific form of project, that being, one employing spatially referenced data. Owing to the set oriented nature of spatial data, do the conclusions determined in this thesis apply equally to projects with non spatially referenced data? Similar work has been done with other types of projects but to what extent can the conclusions be applied generally?

It should also be noted that although the system specified employed spatially referenced data, the system was also highly record oriented. Did the applicability of the Formal Specifications to this system arise due to the record orientation or the spatially referenced

data orientation? This specification was completed using the "Z" formal specification language. Is this particular formal language well suited to record orientation? Repeating this specification using another formal language, such as ZEST or VDM would indicate the effectiveness of the chosen language.

This specification was completed using the product "Formaliser". In order to show whether or not the specification was influenced by the rigors of using this particular product, the specification should also be repeated using another Formal Specification editor and type checker.

In creating the Formal Specification found in Chapter 3, certain decisions had to be made concerning the design and architecture of the system. This fact also indicates that Formal Specifications may be equally useful, if not more so, in other phases, particularly the design phase of a project. An excellent opportunity would present itself to research this area if the system specified within this thesis were to be eventually constructed.

References

Architectural & Civil Engineering Technologies."*GIS Technology*". Southern Alberta Institute of Technology, 1997

Barden. Rosalind & Stepney. Susan & Cooper. David."*Z in Practice*". Prentice Hall. 1994

Bergmann. M. & Moor. J. & Nelson. J.."*The Logic Book*". Random House. 1980

Bjorner. Dines & Jones. Cliff B.."*Formal Specification & Software Development*". Prentice Hall. 1982

Bowen. J. P. & Hinchey. M. G.."*Seven more myths of formal methods?*". IEEE Inc.. 1995

Buckley. F.J.."*Implementing Software Engineering Practices*". Wiley and Sons. 1989

Burrough. P.A.."*Principles of Geographical Information Systems for Land Resources Assessment*". Clarendon Press. Oxford. 1996

DeMarco. T.."*Structured Analysis and System Specifications*". Yourdon Press. 1978

Diller. Antoni.."*An Introduction to Formal Methods*". Wiley and Sons. 1992

Gane. C. & Sarson. T.."*Structured Systems Analysis: Tools and Techniques*". Prentice Hall. 1979

Gause. Donald C. & Weinberg. Gerald M.."*Exploring Requirements - Quality before Design*". Dorset House Publishing, 1989.

Gibbins. P.."*What are formal methods?*". Buttersworth. 1990

Hall. Anthony.."*Seven myths of formal methods?*". IEEE Inc.. 1990

Hayes. Ian."*Specification Case Studies*". Prentice Hall. 1987.

Humphrey. W. S.."*Managing the Software Process*". Addison-Wesley. 1990.

IEEE."*IEEE Guide to Software Requirements Specifications*". IEEE Inc.. 1984

Jacky. Jonathan."*the way of Z*". Cambridge. 1997

King. S. & Hammond. J. & Chapman. R. & Pryor. "*Is Proof More Cost-Effective Than Testing?*". IEEE Transactions on Software Engineering, August. 2000

Laurini, Robert & Thompson, Derek,"*Fundamentals of Spatial Information Systems*", Academic Press, 1996

Logica UK."*Z Specific Formaliser User Guide*", Logica, 1995

Macaulay, Linda A.,"*Requirements Engineering*", Springer-Verlag, 1996.

McKercher, Robert B. & Wolfe, Bertram,"*Understanding Western Canada's Dominion Land Survey System*", University Extension Press, University of Saskatchewan, 1986.

Paulk, Mark C. et al.,"*Key Practices of the Capability Maturity Model*", SEI Technical Publications, 1993.

Pohl, K."*The three dimensions of Requirements Engineering - Fifth International Conference on Advanced Information Systems Engineering*", Springer-Verlag, 1993

Schach, Stephen R.,"*Software Engineering*", Irwin, 1993.

Spivey, J. M.,"*The Z Notation: A Reference Manual - 2nd Edition*", Prentice Hall, 1992.

Stepney, Susan."*Testing as Abstraction*", Logica UK Ltd., 1995

Stepney, Susan."*Personal Communications*", 1998

Taylor, Fraser."*Geographic Information Systems*", Pergamon Press, 1991

Woodman, M.,"*Yourdon dataflow diagrams: a tool for disciplined requirements analysis*" Buttersworth, 1990

Wordsworth, J. B.,"*Software Development with Z*", Addison-Wesley, 1993.

Worrall, Les."*Spatial Analysis and Spatial Policy using Geographic Information Systems*", Belhaven Press, 1991

Yourdon, E. & Constantine, L.,"*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*", Yourdon Press, 1978

Appendix A

1.0 Requirements for a Land Management Information System

This appendix contains a complete narrative specification for a Land Management Information System. This Requirements Specification Document is as accepted by the client and is going to be used to develop the system described within.

1.1 Introduction

1.1.1 Purpose

The following is a Requirements Specification Document (RSD) for a system to manage the Land Information requirements for a major Alberta, Canada, Oil and Gas system. It will be described using a narrative format as prescribed by the client and will approximately follow the IEEE Guide to Software Requirements Specifications (IEEE, 1984). This Requirements Specification Document is as accepted by the client.

1.1.2 Scope

This RSD will describe the required functionality for a Land Management Information System (LMIS). The requirements for this system will be described from the perspective of the 5 user groups, namely:

Construction – This group deals with land issues prior to construction of a pipeline or other facility. Their function is to determine landowners and regulations pertaining to

the affected parcels of land. They must then complete all legal requirements to confirm that construction can begin on the affected land at the required time.

Appraisals – When a right-of-way is required through a parcel of land, obligations are incurred by the company. These obligations are usually, but not always, in the form of monetary compensation. It is the job of the Appraisals group to determine the value of such compensation.

Operations – This user group deals with the day to day issues of land after construction has been completed. This includes, completing all contractual obligations for the ongoing use of the land, maintaining accurate information in regards to the land and responding to all inquiries with regards to additional uses of the land.

Land Agents – Land Agents are the people that communicate directly with the landowners. They gather information, resolve disputes and generally maintain a good rapport between the landowner or occupant and the Company.

Legal – The legal department ensures that all regulations and obligations between the landowner, company and government are properly completed.

1.1.3 Definitions, Acronyms and Abbreviations

CLI – Canada Land Index – A numeric value used to designate what a particular parcel of land can be used for.

LLD – Legal Land Description – The description of a parcel of land as registered with the government of Alberta Land Registry Information System.

LMIS – Land Management Information System – A system used to maintain information about land including landowners, occupants, uses, regulations and obligations.

LRIS – Land Registry Information System – A computer application operated by the government of Alberta to provide land registry data to the public.

RSD – Requirements Specification Document – A document describing the requirements for a system in sufficient detail that the actual system can be designed and constructed from the document.

1.2 General Description

1.2.1 Product Perspective

This product provides information about parcels of land and allows land management activities. It must be available 24 hours a day, 7 days a week. It must provide an on-line, visual representation of all land parcels within the province of Alberta on which the company has an obligation. By obligation, it is meant that the company either owns the land, leases the land, has a right-of-way through the land or has an obligation to the landowner or occupant due to having other obligations on adjacent parcels of land.

1.2.2 Product Functions

Product functions are subdivided by the 5 different user groups. This section provides an overview of the functions of each of the groups.

Construction – It is the function of the Construction Group to determine the affected legal descriptions for any parcel of land for any new construction project. Once

the legal descriptions for all affected parcels have been determined, the Construction Group must then determine all legal obligations that the company has towards any landowners, occupants, municipal governments and the provincial government. This is done in conjunction with the Legal Department. Once obligations have been determined, the Construction Group notifies the Appraisal Group of the affected parcels who determine fair compensation for any encumbrances. This information is given to the Land Agents who visit the Landowners or Occupants, explain the nature of the new project and attempt to obtain written legal consent.

Once consent has been given, all legal documents must be registered with the government and all pertinent data entered into the system.

Appraisals – It is the function of the Appraisals Group to determine fair compensation for the right to have a right-of-way on a persons land. This can be done at construction time for a new project, during arbitration for a disputed claim or when a contract comes up for renewal. The Appraisals Group must be able to determine, through the system, the affected parcels, all data in regards to a particular parcel that would affect its value and be able to update the system with the new data.

Operations – The primary function of the Operations Group is to fulfill the company obligations to the landowner, occupant or other affected party. This includes payment of compensation as detailed in the contract, notification to the landowner or occupant of any new work to take place and renegotiation of the contract upon expiry. The Operations Group must also maintain up to date landowner and occupant data.

Usually a single parcel can have several rights-of-way. It is also the function of the Operations group to work with other companies in negotiating and granting permissions for one right-of-way to cross or encroach on another.

Land Agents – Land Agents are the main communication vehicle between the Company and the Landowner and/or Occupant. They use the system to obtain contact information and to store contact history. They deliver legal documentation which is output from the system. and maintain status information on the different landowners. occupants or any other affected party.

Legal – All legal documentation such as contracts. caveats. rights-of-way agreements etc. are output from the system. The Legal Department determines which documents are applicable to which specific situation. They must therefore be able to maintain all potential documents within the system and specify which ones are to be used in each instance. This must be on-line and immediate.

1.2. User Characteristics

Each of the 5 different user groups has unique characteristics and so will be listed here separately.

Construction – The Construction Group work in the office. not in the field. The system must. therefore. be available on their desktops. in their offices with local access to printers. External access must be available to the Alberta Government online registry system for acquiring current land title data and electronic submission and filing of legal documentation.

Appraisals – The Appraisal Group function in much the same way as the Construction Group. They require the same access as the Construction Group plus additional access to topographical information, government agricultural databases, including the Canada Land Index, and current real estate information.

Land Agents – Land Agents work both in the office and in the field. They require on-line access from both locations as they must keep accurate and extensive field notes. They must have up to date access to all contact information as well as contact history.

Legal – The Legal Department requires on-line office access only. Their function is to maintain legal documentation and determine which documentation is appropriate in any given situation. Since they do not file or register forms directly, they do not require on-line access to any external government databases. Government regulations are currently communicated by mail but enhancements to the system to allow email of new policies and circulars by the government should be anticipated.

1.3 Specific Requirements

1.3.1 Functional Requirements

Functional requirements will be divided into 6 categories. The first category will be general in that they will be requirements which are appropriate to all users. They are user independent. The remaining 5 sections will consist of the requirements for each specific user group.

1.3.1.1 General Functional Requirements

1.3.1.1.1 General Requirement 1 – Ease of Use

GR1 – The system must be easy to use. There are manual methods to accomplish almost all functions which will be incorporated into this system and the system must be easier to use then the manual methods in order to realize the benefits.

1.3.1.1.2 General Requirement 2 – Y2K Compliant

GR2 – The system must be Year 2000 compliant. Even if the system is not delivered until after the year 2000 it will contain historical data. 4 digit years are required for all dates.

1.3.1.1.3 General Requirement 3 - Secure

GR3 – The system must be secure. All users are required to have a user id and password in order to gain access. Access to each of the different functions is to be individually controlled. There is to be a single security administrator function which will control access.

1.3.1.2 Construction Functional Requirements

The following Activity Chart shows the Construction function of this LMIS system.

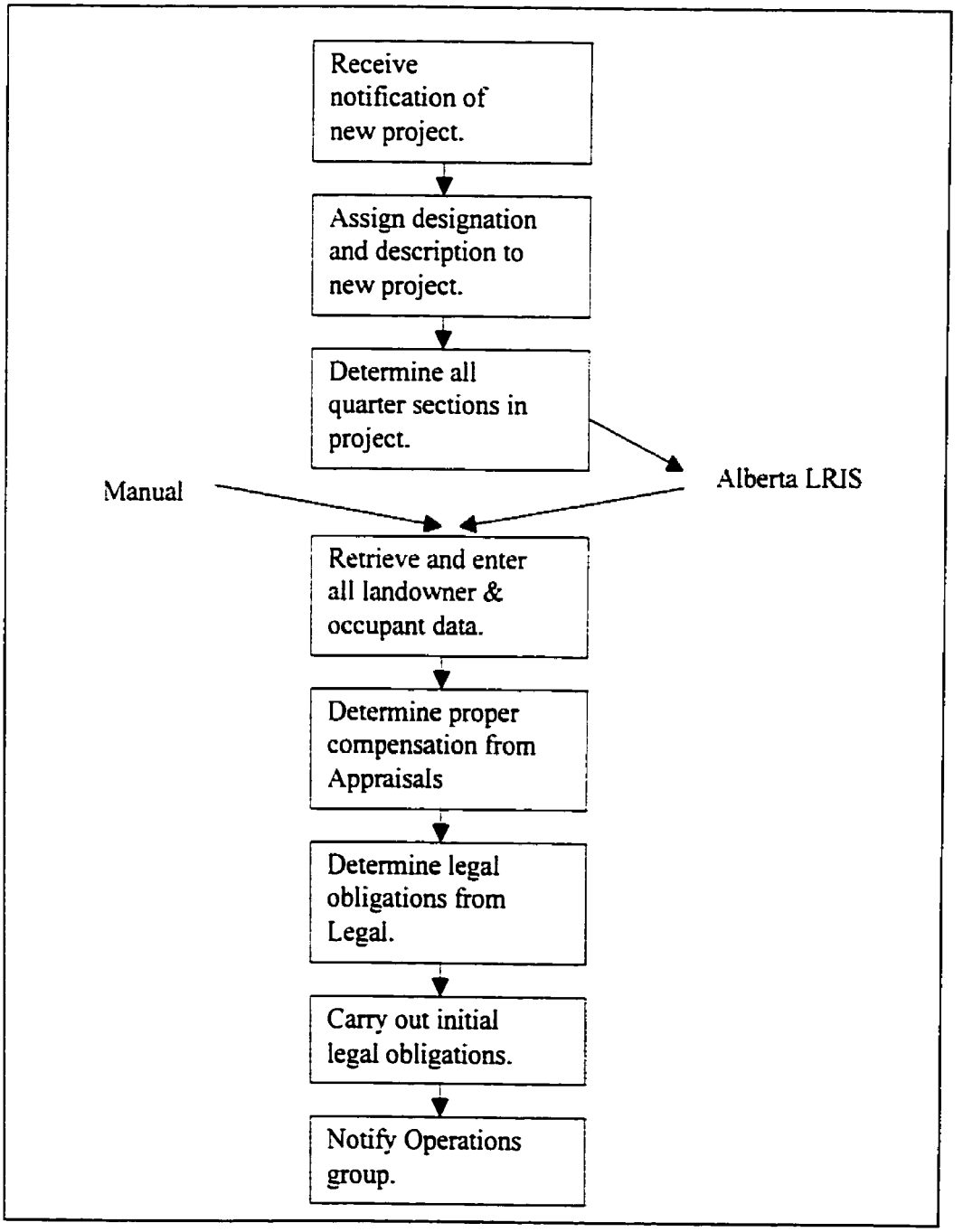


Figure A.1 – Construction Activity

1.3.1.2.1 Construction Requirement 1 – Add Project

CR1 – When the Construction Group is notified that a new project is being considered they must be able to designate a name and number for that project. All relevant project information can be obtained using that unique identification.

1.3.1.2.2 Construction Requirement 2 – Assign Quartersections

CR2 – All projects are associated with a parcel or parcels of land. From construction drawings prepared by the Engineering department, the Construction Group can determine all affected quarter sections. They must have the ability to create a list associating all affected quarter sections with the project. Affected quarter sections may be land on which construction will actually take place or land adjacent to the construction that contains a Right-of-Way.

1.3.1.2.3 Construction Requirement 3 – Assign Subdivisions

CR3 – The construction department must have the ability to determine all legal subdivisions within all affected quarter sections. This is accomplished by providing the Alberta Land Registry Office with a list of the affected quarter sections. This list is given electronically through the Land Registry Information System (LRIS), which is a system operated by the Alberta Government for use by the public.

1.3.1.2.4 Construction Requirement 4 – Receive landowners

CR4 – The LRIS system returns a list of all legal descriptions and landowners within all requested quarter sections. This list is electronic and can be received either by mailed diskette or email.

1.3.1.2.5 Construction Requirement 5 – Parse LRIS List

CR5 – The system must provide for the ability to parse the returned LRIS list and record all legal land description and landowner information for all affected quarter sections.

1.3.1.2.6 Construction Requirement 6 – Record Land Data

CR6 – The system must provide for the capability to record the following data for each legal land description affected:

Legal Land Description (LLD)

Project Identification

Landowner Name

Landowner Address

Landowner contact information

Phone (Home. Business. Fax. Cellular)

Email

Occupant (if other than landowner)

Occupant Address

Occupant contact information

Phone (Home. Business. Fax. Cellular)

Email

Total land area in Hectares

Canada Land Index (showing land usage)

Date land last sold

Last selling price

1.3.1.2.7 Construction Requirement 7 – Manually enter land data

CR7 – Not all of the above information is always on the LRIS report. A facility must exist to allow the Construction agent the ability to manually enter data as it becomes available.

1.3.1.2.8 Construction Requirement 8 – Provide Audit Trail

CR8 – An Audit trail must be provided so that the agent who entered the information can be determined as well as the date and time it was recorded.

1.3.1.2.9 Construction Requirement 9 – Notify Appraisals

CR9 – When the construction agent has determined that all affected LLD's have been determined, as well as all landowners, he/she must be able to electronically pass the list to the Appraisal Group for determination of fair compensation. This list must contain all affected LLD's, all landowners and a detailed description of the project, including a link to the Engineering drawings if possible. The appraiser will enter the suggested compensation associated with the LLD and Project Id.

1.3.1.2.10 Construction Requirement 10 – Notify Construction Group

CR10 – Once the appraiser has determined and updated the data with suggested compensation, the construction agent must be electronically notified that this task is complete and that they can proceed with their next task.

1.3.1.2.11 Construction Requirement 11 – Select Forms

CR11 – The Construction agents and the Legal Department must have the ability to select from a list of legal forms which form(s) is to be sent to a specific landowner or occupant.

1.3.1.2.12 Construction Requirement 12 – Package Forms

CR12 – These forms must be able to be grouped into "packages" so that if a list of multiple forms is always sent for a specific project type then the package can be selected for creation as opposed to multiple individual forms.

1.3.1.2.13 Construction Requirement 13 – Designate Landowners

CR13– The Construction Group and/or Legal Department must have the ability to designate non landowners or occupants to be the recipient of documentation. For example, local governments, municipal governments, Provincial government, Royal Canadian Mounted Police and local newspapers.

1.3.1.2.14 Construction Requirement 14 – Enter Data en-mass.

CR14 – The Construction Group must be able to enter contract start dates, contract terms and end dates for each landowner either individually or en-mass. Usually all contracts for a given project have the same start and end dates but if a specific landowner seeks arbitration then that particular contract can have different dates and terms.

1.3.1.2.15 Construction Requirement 15 – Notify Operations Group

CR15 – A method for the Construction Group must exist so that they can declare a project ready for construction and can pass it to the Operations Group.

1.3.1.3 Appraisal Functional Requirements

The following Activity Chart shows the Appraisal function of this LMIS system.

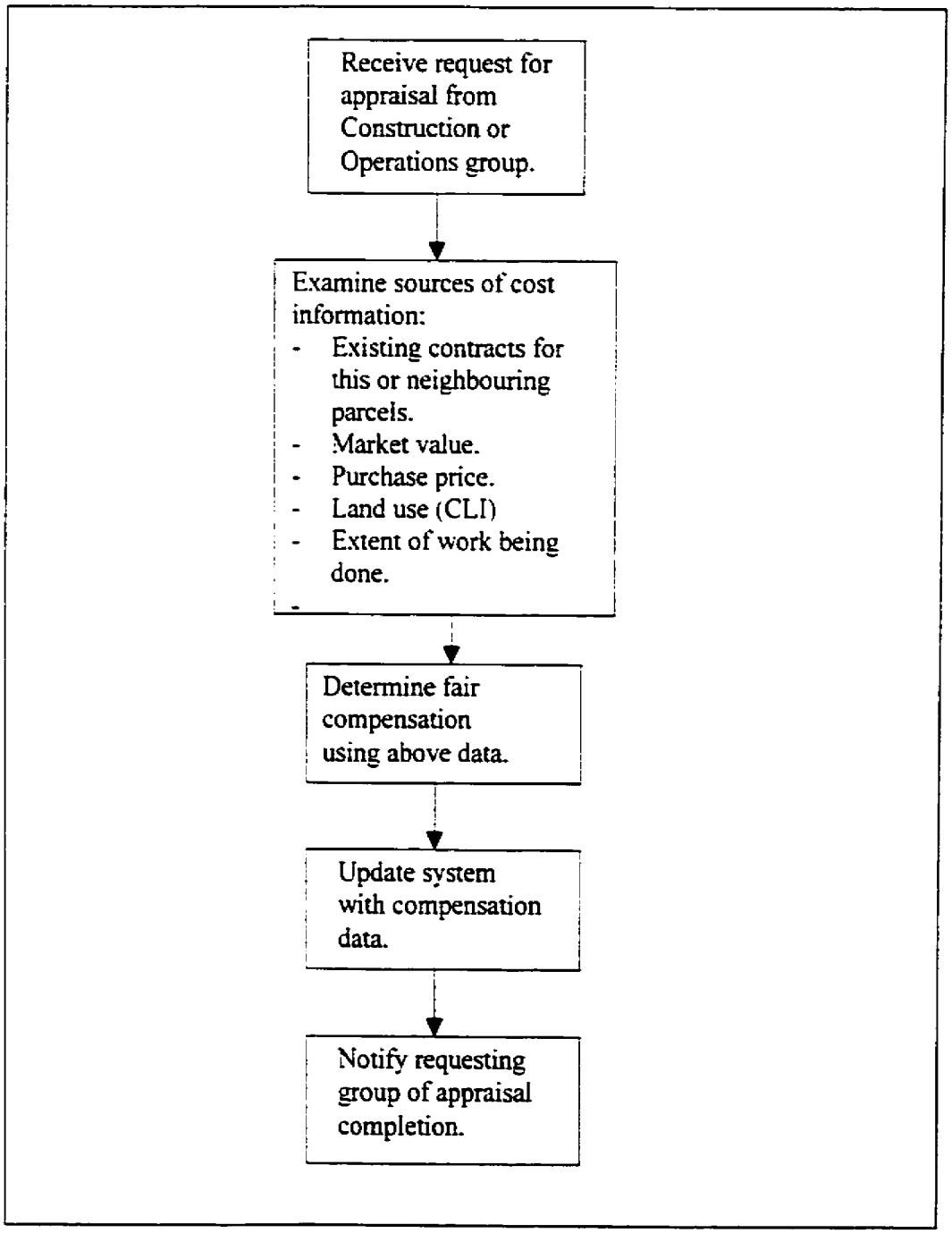


Figure A.2 – Appraisal Activity

1.3.1.3.1 Appraisal Requirement 1 – Notify Appraisals

AR1 – A method must exist to allow the Construction Group or Operations Group to inform the Appraisal Group that an appraisal is required. This appraisal can be requested at the beginning of a new construction project or at the renewal of an existing contract for a completed project.

1.3.1.3.2 Appraisal Requirement 2 – Access to Data

AR2 – The appraiser must have ready access to all the information entered by the construction agent. Land size. Right-of-Way width, project particulars and land use, all contribute to the final amount of compensation determined by the appraiser.

1.3.1.3.3 Appraisal Requirement 3 – Assign Compensation

AR3 – The appraiser must be able to associate a compensation amount with a particular LLD and project. Multiple projects may be on the same parcel of land.

1.3.1.3.4 Appraisal Requirement 4 – Mass Update

AR4 – Usually most parcels within a specific township will receive the same amount of compensation per hectare. A method must therefore exist for mass update. Entering an amount per hectare and selecting the affected LLD's from a list should update all parcels.

1.3.1.3.5 Appraisal Requirement 5 – Notify Requesting Group

AR5 – The Appraiser must have a method of informing the requesting group, either Construction or Operations, that the appraisal is complete and the requesting group can proceed with their next task.

1.3.1.4 Operations Functional Requirements

The following Activity Chart shows the Operations function of this LMIS system.

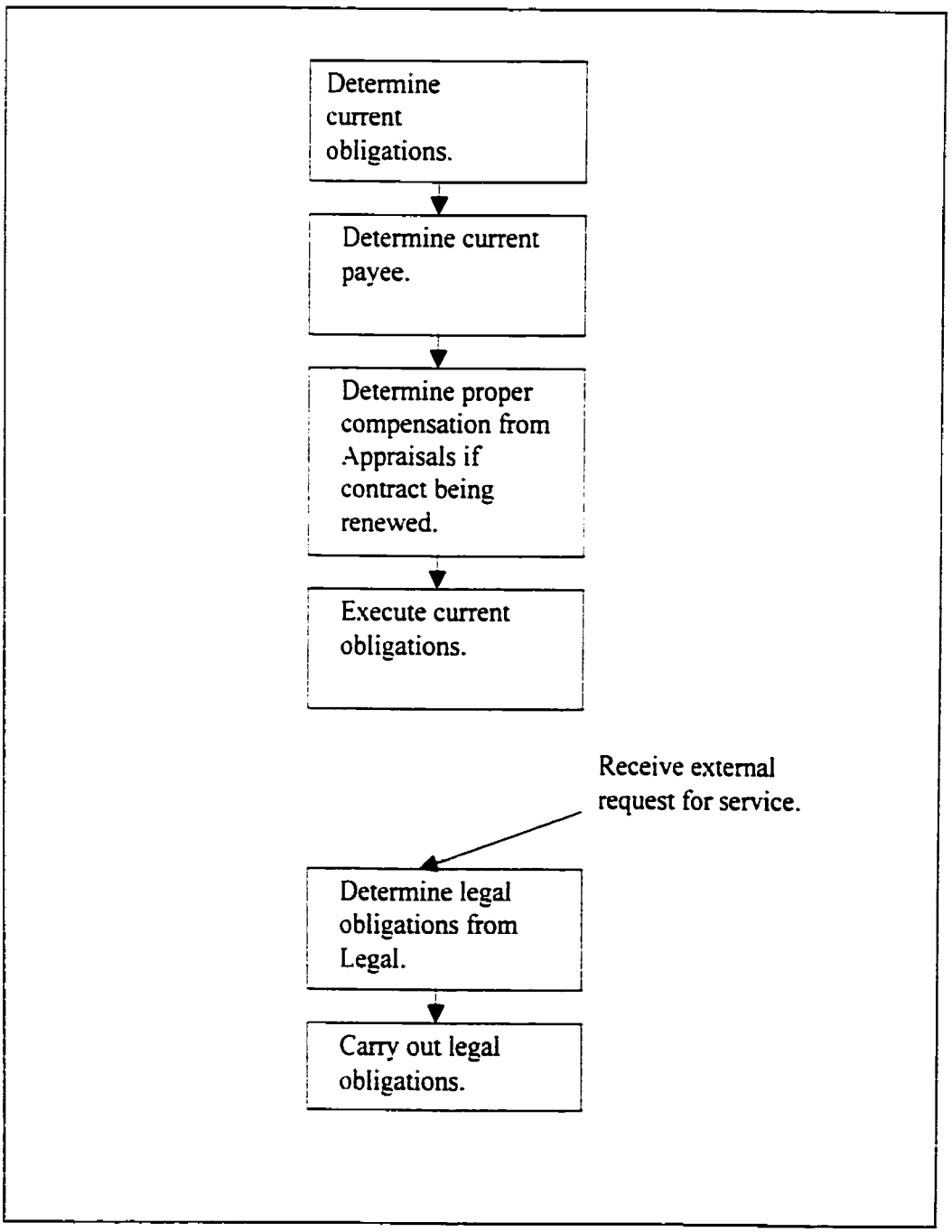


Figure A.3 – Operations Activity

1.3.1.4.1 Operations Requirement 1 – Retrieve Obligations

OR1 – It is one of the jobs of the Operations department to satisfy contractual obligations. They must therefore have notification of upcoming obligations. This should be in the form of a hard copy report produced on a daily basis.

1.3.1.4.2 Operations Requirement 2 – Get Obligation Data

OR2 – Obligations involve paying specific amounts of money to specific people at specific times. Each of these 3 items is determined by contract.

1.3.1.4.3 Operations Requirement 3 – Update Landowner Data

OR3 – Obligations usually involve paying sums to the current landowner or occupant. The Operations Group therefore require access to the identical functionality as the Construction Group in determining and updating current data.

1.3.1.4.4 Operations Requirement 4 – Notify Appraisal Group

OR4 – The Operations Group must have the ability to notify the Appraisal Group at contract renewal time to determine a revised compensation. They must also have the ability to be notified by the Appraisal Group when the appraisal is complete and updated.

1.3.1.4.5 Operations Requirement 5 – Print Documentation

OR5 – The Operations Group must have the ability to create documents as designated by the Legal Department in the same fashion as the Construction Group.

1.3.1.4.6 Operations Requirement 6 – Record Encroachment Requests

OR6 – Requests are received from third parties to encroach on an existing Right-of-Way for several reasons. The Operations Group must have the ability to record all requests

identifying the third party, date, Operations Agent and request including LLD's affected and the nature of the request.

1.3.1.4.7 Operations Requirement 7 – Create Custom Forms

OR7 – In addition to the legal documentation concerning the current project, the Operations Group must also have the ability to create both form and customized letters and contracts to third parties concerning their requests.

1.3.1.4.8 Operations Requirement 8 – Register Encroachments

OR8 – The Operations Group must have the ability to register encroachment contracts with the provincial government in the same fashion as is done with construction contracts.

1.3.1.5 Land Agents Functional Requirements

The following Activity Chart shows the Land Agents function of this LMIS system.

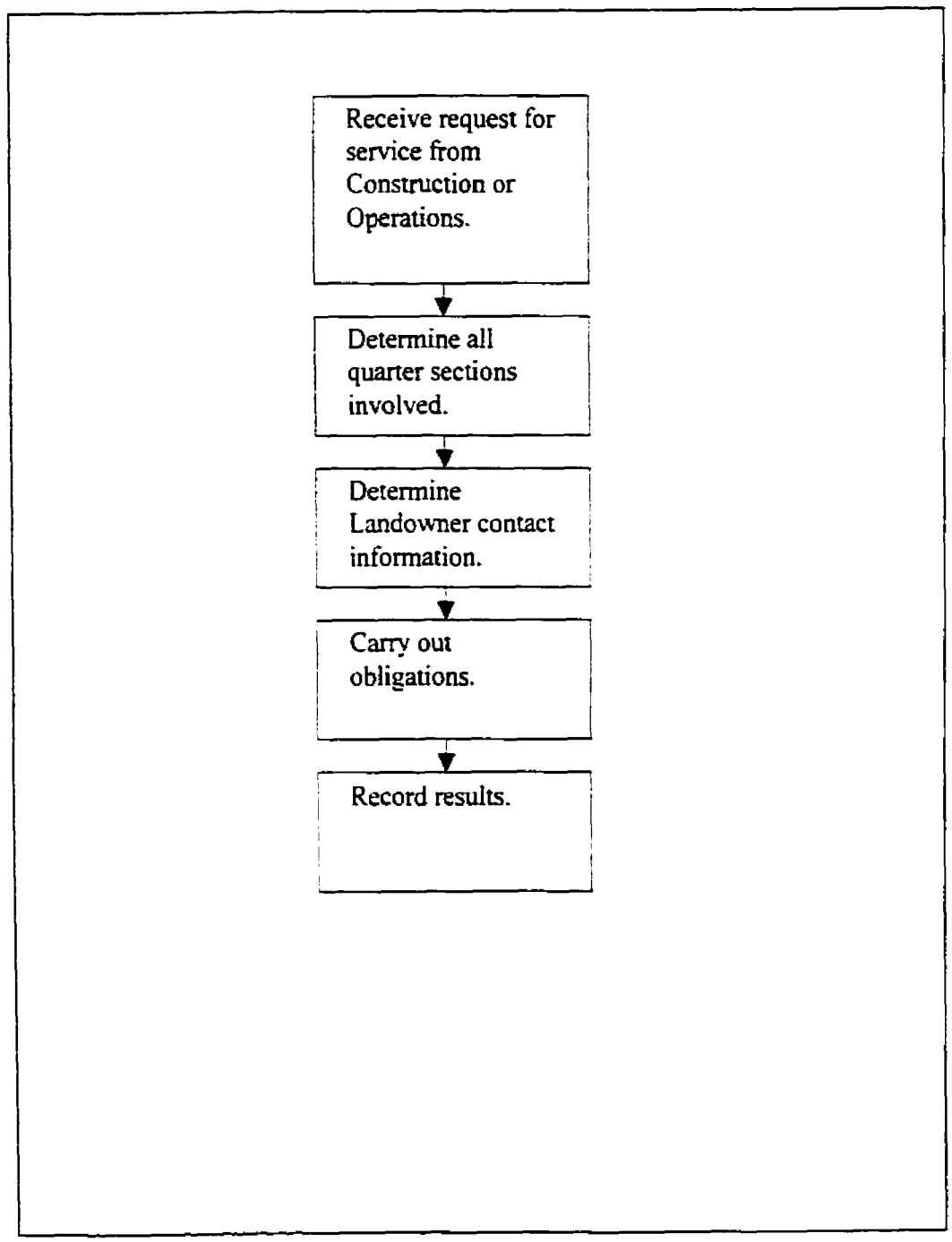


Figure A.4 – Land Agent Activity

1.3.1.5.1 Land Agent Requirement 1 – Access Contact Data

LAR1 – The Land Agent uses the system to determine and record contact data. The first form of contact is usually notification of an upcoming project on a person's land. They therefore, must have access to project particulars and landowner/occupant data.

1.3.1.5.2 Land Agent Requirement 2 – Access from Remote Locations

LAR2 – Land Agents works in the field. They must have access to the data they require while at remote locations.

1.3.1.5.3 Land Agent Requirement 3 – Record New Contact Data

LAR3 – The Land Agent must be able to record contact history. The system requires the ability for them to record unlimited free text in regards to contact. This data must be keyed by project id and date.

1.3.1.5.4 Land Agent Requirement 4 – Acknowledge Documentation

LAR4 – One of the Land Agents tasks is to deliver legal documentation. They must have the ability to enter into the system that each piece of documentation has been delivered and accepted by the recipient. Each document required for each LLD must then be separately identified with the Land Agent being able to confirm its delivery.

1.3.1.6 Legal Functional Requirements

The following Activity Chart shows the Legal function of this LMIS system.

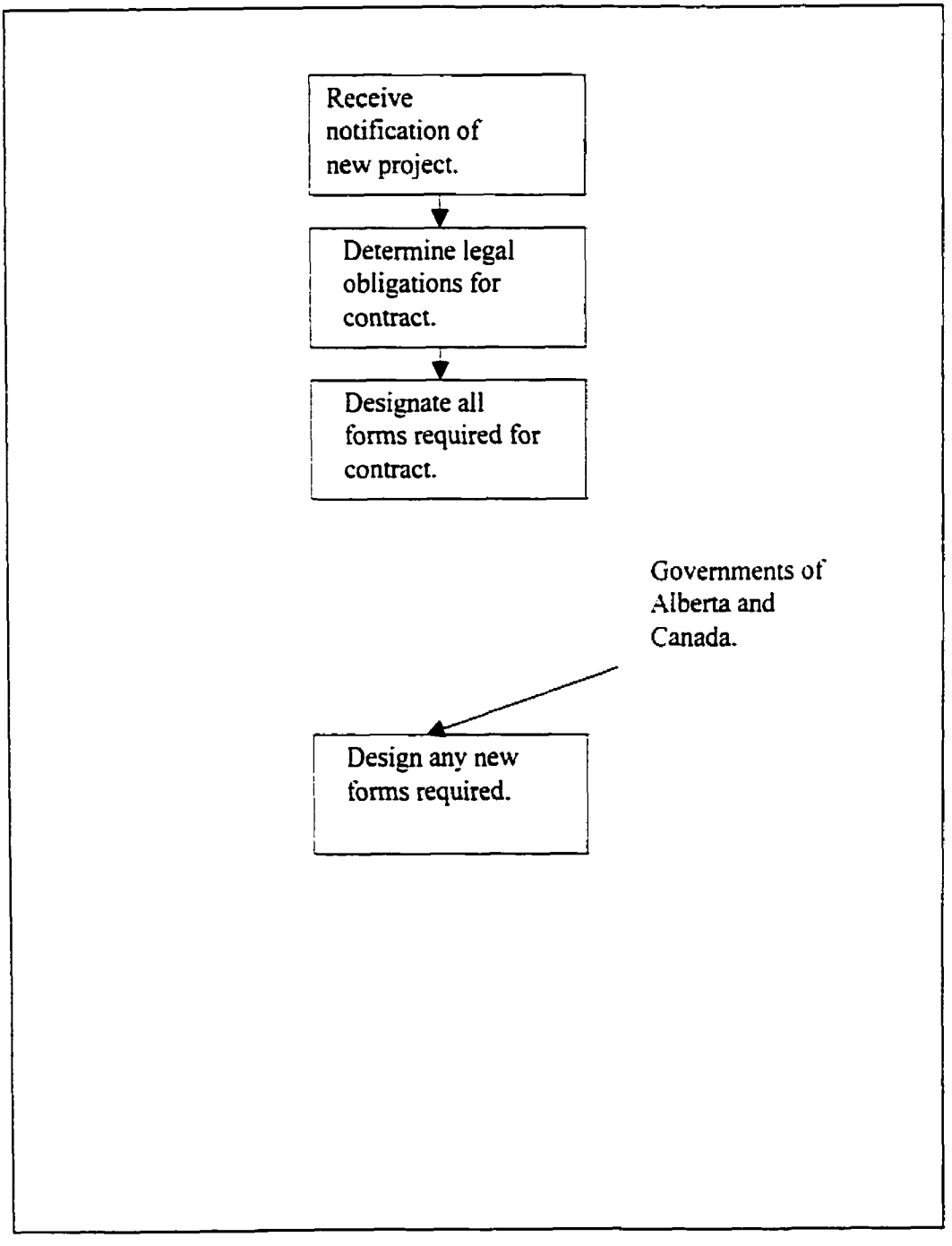


Figure A.5 – Legal Activity

1.3.1.6.1 Legal Requirement 1 – Notification of Project

LR1 – The Legal Group confirms that all legal requirements are properly met for any project, new or existing. They therefore must have the ability to be electronically notified of a project by either the Construction group or the Operations group.

1.3.1.6.2 Legal Requirement 2 – Access to Government Data

LR2 – Legal requirements consist of the proper preparation, delivery and registration of all legal documents as prescribed by the Province of Alberta and the Government of Canada. They must therefore have the ability to determine such requirements in the manner prescribed by the respective government agencies be this manual or electronic.

1.3.1.6.3 Legal Requirement 3 – Designate Documentation

LR3 – Once the legal requirements for a specific project are known, the Legal group must have the ability to designate what forms must be completed, delivered and registered for each parcel of land.

1.3.1.6.4 Legal Requirement 4 – Direct Documentation

LR4 – There can be multiple recipients for documentation pertaining to an individual LLD. Some documents may be intended for the landowner, some for the occupant only if different from the landowner. The Legal group then must have the ability to direct documents to specific individuals or groups.

1.3.1.6.5 Legal Requirement 5 – Third Party Documentation

LR5 – Documents may not always be intended for or associated with a specific LLD. Some documents may be intended for local newspapers, municipal governments, owners or occupants of land adjacent to the project etc. The Legal group therefore requires the ability

to designate some documents to third parties and have their deliveries and registrations confirmed in a manner similar to documents intended for landowners/occupants.

1.3.1.6.6 Legal Requirement 6 – Maintain Document List

LR6 – All available documents must be selected from a list. The legal group maintains this list having the ability to add, delete or modify document names and identifications.

1.3.1.6.7 Legal Requirement 7 – Package Documentation

LR7 – The Legal group must have the ability to designate individual documents for delivery or pre-determined groups. These would consist of packages of documents which all would be delivered together under specific circumstances.

1.3.1.6.8 Legal Requirement 8 – Create Templates

LR8 – The Legal group determines the format of each and every document that can be delivered. They must therefore have the ability to create templates for all required documents. These templates must have the ability to designate what is fixed text and what can be filled in by the Construction group or Operations group. The Microsoft Word Mail Merge facility or the Jetform product both have this capability.

1.3.1.6.9 Legal Requirement 9 – Determine Data Source for Templates

LR9 – Document templates must have the ability to determine sources of data for areas designated as variable by the Legal Group.

1.3.2 External Interface Requirements

Requirements should designate *what* is required in any particular system, not *how* they are to implemented. The following section deals with requirements for the Human-Computer Interface (HCI) as well as interfaces to external systems. Some of these may be considered *how* in that they designate a specific method of implementing functionality. These may be mandated due to a need to adhere to Corporate Standards, follow a prescribed procedure or make use of existing technology within a company.

1.3.2.1 External Interface requirement 1 – Windows NT 4.0

EIR1 – The system must be able to run on a Windows NT 4.0 platform.

1.3.2.2 External Interface requirement 2 – Seamless Access to LMIS

EIR2 – Access to the Province of Alberta Land Registry Information System must be seamless. That is, access must be from the same computer as this LMIS is running on with files being created by the LRIS system which are accessible to the LMIS.

1.3.2.3 External Interface requirement 3 – Access to Autodesk Mapguide

EIR3 – The Land Management Information System is not a Geographical Information System (GIS) in that it does not contain all the information contained in many of the commercially available GIS's, such as Mineral data, Oil & Gas data etc.. however a visual component is required. The company has purchased the StrataWeb product (http://www.strataweb.com) which uses the Autodesk Mapguide drawing engine. (http://www.autodesk.com/products/mapguide)

1.3.2.4 External Interface requirement 4 – Seemless Access to StrataWeb

EIR4 – StrataWeb must be accessible from within the LMIS. The user must not have to exit the LMIS to access StrataWeb and then re-establish a connection the LMIS.

1.3.2.5 External Interface requirement 5 – StrataWeb Data

EIR5 – The following data are required within StrataWeb:

Map of Alberta

All cities and towns

All township lines drawn

All range lines drawn

All registered pipelines

All lakes, rivers and streams

Canada Land Index value for each quarter section in the province

1.3.3 Performance Requirements

1.3.3.1 Performance requirement 1 – Response Time

PR1 – No response time for any local activity is to exceed 3 seconds. Responses from external systems (i.e. LRIS) are the only exceptions.

Appendices

The document as was signed off by the client contained in its appendices technical data on Jetform, StrataWeb, Autodesk Mapguide and Microsoft Windows Mail Merge. These will be omitted from this thesis.