An End-User Intelligible Specification Language and Its Execution

DOCTORAL THESIS

By

Pathirage Gamini Wijayarathna

This dissertation submitted to the Department of Information Systems Design, Graduate School of Information Systems, The University of Electro-Communications in partial fulfillment of the requirements for degree of Doctor of Engineering.

Department of Information Systems Design
Graduate School of Information Systems
The University of Electro-Communications
1-5-1 Chofugaoka
Chofu-Shi, Tokyo, 182, Japan
email : gamini@maekawa.is.uec.ac.jp

September 1998

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

This chapter contains the background and objectives of this research. It discusses the importance of specifying requirements in software developments, importance of representing incomplete relative temporal knowledge, present approaches to both specification languages and temporal knowledge representation, and also execution of software requirements specifications. Finally it provides the objectives of the research.

1.1 Background

Software developments commence with identifying, specifying and documenting requirements for an information system. Software requirements specification says, what the final system needs to perform, instead of how it should be done[Borgida 1985]. This is the most crucial and difficult step in information system development process. Accuracy of software requirements specification is largely responsible for the low cost and on-schedule completion of information system development projects[Vessey 1994].

Writing of software requirements specifications needs a requirements specification language. A variety of software requirements specification languages are available in the literature. They can be classified into two categories: formal and informal requirements specification languages. Formal requirements specifications have sound mathematical basis and use formal notation to produce concise, clear and precise software requirements specifications; whereas informal requirements specification languages employ graphics and semiformal textual grammars to provide end-user comprehensible software requirements specifications[Fraser 1991].

Intelligibility of the software requirements specification provides better communication between end-users and developers. Communication barrier between end-user and developer is one of the unresolved problems in requirements engineering[Hsia 1993].

Most formal requirements specification languages do not support user friendly representation. Hence Hall[Hall 1990] suggests that the formal requirements specification must be accompanied by a natural language description. Recent requirements specification languages employ an object-oriented approach to document requirements for an information system[Mylopoulos 1990][Jungclaus 1996]. A study shows that the object methodology is difficult to understand even by the novice systems analyst[Vessey 1994]. Hence it is obvious that the non-technical end-user will experience more difficulties with object-oriented representations. Although object-oriented techniques are well established in system designing and coding, its contribution to requirements engineering is not so apparent[Hsia 1993]. However, object-oriented principles (abstraction: aggregation & specialization, inheritance, complex objects) are helpful when dealing with complex systems[Sigfried 1996]. Therefore, the systems analyst will benefit from an object-oriented conceptual modeling technique.

Representing and reasoning about temporal knowledge play an important role in information systems developments. We need to refer to what has happened already, what is happening now and what will happen in the future and then to make decisions based on them[Torsun 1995]. Situation calculus, Event calculus and First-Order Temporal logic (FTL) have been used for these purposes[Russell 1995]. FTL has temporal operators in addition to the logical connectives and quantifiers in first-order logic. Allen[Allen 1984] suggests thirteen possible temporal relationships. However, all the FTL temporal operators available in the literature[Tuzhilin 1995][Torsun 1995][Abadi 1990][Manna 1995] are not capable of representing Allen's[Allen 1984] temporal relationships and all of them use the current state (now) as the origin in time line. Torsun[Torsun 1995] says that FTL is not complete as a first-order logic. In Templar [Tuzhilin 1995], temporal predicates can have temporal operators such as **always_in_the_future**, **sometimes_in_the_future**, **for_time**, **within_time**. However relative temporal knowledge has been represented using the clauses **before**, **while** and **after**. TROLL[Jungclaus 1996] employs existing temporal operators and the predicates "before" and "after". TELOS[Mylopoulos 1990] uses a slightly modified version of Allen's [Allen 1984] temporal relations to represent temporal knowledge. These show the importance of incomplete relative temporal knowledge in software requirements specifications.

Once the requirements specification is ready, one has to consider how to implement the specification correctly. Therefore the program verification becomes very important. Temporal logic has been used for program verifications[Manna 1995]. However, software requirements specification languages based on temporal logic can produce an executable specification making the verification process easier and faster in automated program generations since the specification language itself can be used as a programming language[Barringer 1996]. As discussed, it is hard to find a requirements specification language that satisfies all the criteria (end-user understandable, executable specification which supports object-oriented principles and is capable to represent incomplete relative temporal information).

1.2 Objective

Since incomplete relative temporal knowledge play an important role in information systems, the proposed language should be capable of representing them in end user understandable form and should be flexible enough to introduce user defined temporal relations to the language. If not, systems analysts and end users have to use a predefined representation scheme which restricts the expressiveness of the language. On the other hand, object-oriented modeling techniques are capable to handle complex systems easily. Therefore the proposed language should provide facilities for object-oriented modeling techniques. Hence the objective of this research is to design a specification language which can be used for object oriented modeling and further it provides executable, intelligible specifications.

This thesis can mainly be divided into three parts. First part discusses conventional approaches to represent incomplete relative temporal knowledge and introduces new logical connective TAND. The second part describes the development of a new temporal logic called TANDTL which employs TAND connective to represent temporal knowledge. The final part explains the features of new specification language called GSL which is based on the temporal logic TANDTL. GSL is executable and end-user intelligible. As discussed above, the proposed specification language should be able to provide object-oriented modeling. Therefore it is necessary to have a model of a system and examples to introduce the features of the language. Chapter 2 provides the model of a system adopted in this research and the pilot system which is going to be used to provide example throughout this thesis. Further the specification language GSL is designed giving special consideration to accounting information systems. Therefore Chapter 2 discusses about accounting information systems, its users, its subsystems and accounting language. Chapter 3 provides a detailed discussion about representing incomplete relative temporal information. This chapter can be considered as the main chapter of this thesis, as it provides the basis for the remaining chapters. It discusses about conventional temporal knowledge representation methods, problems with them and introduces a new logical connective TAND to represent incomplete relative temporal knowledge. Further it shows how TAND can be used to represent temporal knowledge in information systems.

One of the objectives of this research is to develop an executable specification language based on temporal logic. Chapter 4 introduces the conventional temporal logic, their problems and proposes new temporal logic called TANDTL which removes all the existing temporal operators and uses only TAND connective to represent temporal knowledge. Chapter 5 discusses the similarities between logic and the view of a system adopted in this research and it shows how logic TANDTL can be executed. The purpose of this chapter is to show that a specification language based on TANDTL can be executed since TANDTL is executable. The remaining is to introduce the specification language GSL. Chapter 6 discusses about present specification languages and introduces GSL in detail. Chapter 7 provides the discussion and Chapter 8 concludes the thesis. Appendices A and B provide an application of GSL to real problems.

CHAPTER II

ACCOUNTING INFORMATION SYSTEMS

This chapter introduces the accounting information systems which is the main application area of this research. It introduces what the accounting system is, its users, its subsystems and accounting language. Then it introduces the pilot system which is going to be used to provide examples. It is a kind of a subsystem of an accounting system of a travel agent who provides air line reservations to passengers. Then it discusses about requirements of an information system. Finally it presents the view of a system adopted in this research to model accounting systems and its subsystems. It further discusses why this particular view of a system is adopted and how it can solve the problems of other views of systems.

2.1 Accounting systems

2.1.1 Accounting as an information system

Accounting is an information system that measures, processes and communicates financial information about an identifiable economic entity[Needles 1993]. Bookkeeping is not accounting but it is an important part of accounting. Bookkeeping is the process of recording financial transactions and keeping financial records. The modern accountant is concerned not only with record keeping but also with a whole range of activities that involve planning and problem solving; control and attention directing; and evaluation, review, and auditing. Accounting is generally known as the "language of business"[Eisen 1994]. According to the definition accounting has three major functions:

1. It measures business activities by recording data for future use.
2. The data are stored until needed and then processed to become useful information.
3. The information is communicated, through reports, to the decision makers.

Figure 2.1 shows the process of a typical accounting information system. With the widespread use of the computers today, many of the information needs are being organized into a management information system (MIS). Accounting systems is the financial hub of the management information system. The management information systems consist of many inter connected subsystems including accounting system. Therefore accounting systems interact with many non-financial systems in order to provide the required information. These non-financial systems are represented in business activities in Figure 2.1.

2.1.2 Users of an accounting system

The users of an accounting system can be divided into three:

1. Those who manage a business,
2. Outside business enterprises who have a direct financial interest in the business, and
3. People, organizations, and agencies that have an indirect financial interest in the business.

These groups are shown in Figure 2.2. Behavior of these user groups are affected to the overall management information system.

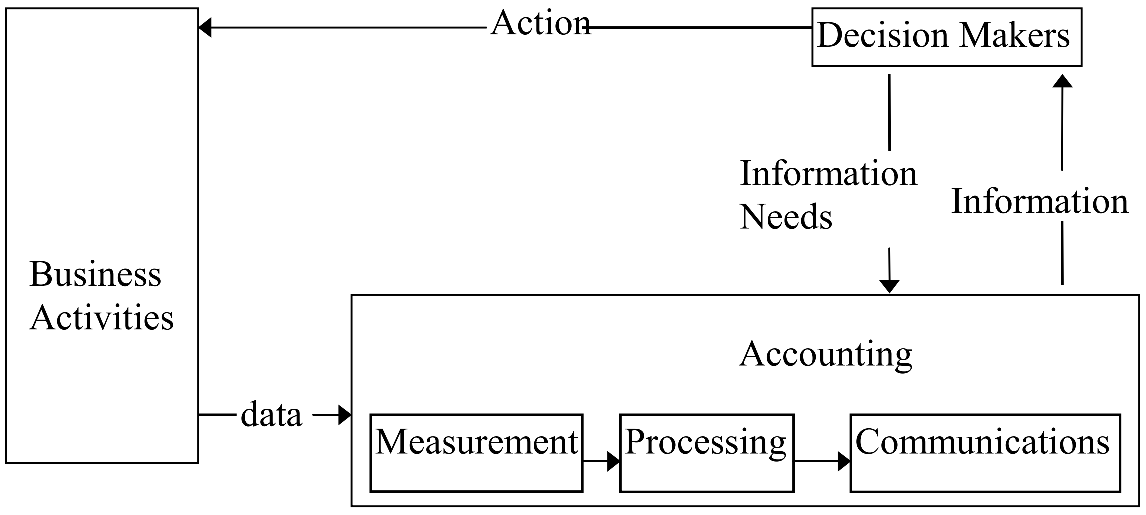


Figure 2.1 Accounting as an information system

Management

Management is the group of people who have overall responsibility for operating a business and for meeting the company's goals. Success and survival in a competitive business environment require that management concentrate much of its effort on two important goals: profitability and liquidity. Managers must decide what to do, how to do it, and whether the results match their original plans. These decisions are based on analysis of accounting information. Therefore management is one of the most important users of accounting information. Their decisions may include modifications to the accounting systems.

Users with direct financial interests

Investors and potential investors are interested in the past performance of the company. Potential investors need accounting information before they invest on the business while those who already invested are interested to know about the success of the business. Many companies borrow money from other organizations. Creditors, those who lend money or deliver goods or services to company on credit basis are mainly interested to know whether the company is in a position to pay back their credits. Potential creditors evaluate financial stability of the company before they lend money or services. Modifications to the existing accounting system may be required to incorporate the creditors decisions.

Users with indirect financial interests

Society as a whole (such as government and public groups) is the most important user of accounting information. Government policy changes will effect an accounting system. Labor unions, employees are also important user groups. Their behavior will bring modifications to the system. These changes will come with short notice. Sometimes it may be few days or few hours. Therefore an accounting system should be flexible enough to deal with sudden changes to the system.

2.1.3 Accounting systems and its subsystems

An accounting system itself has several subsystems. Components of typical accounting system are given in Figure 2.3. The concept of subsystems is applied through various special purpose journals and subsidiary ledgers. For example, there can be Sales Journal, Purchase Journal,

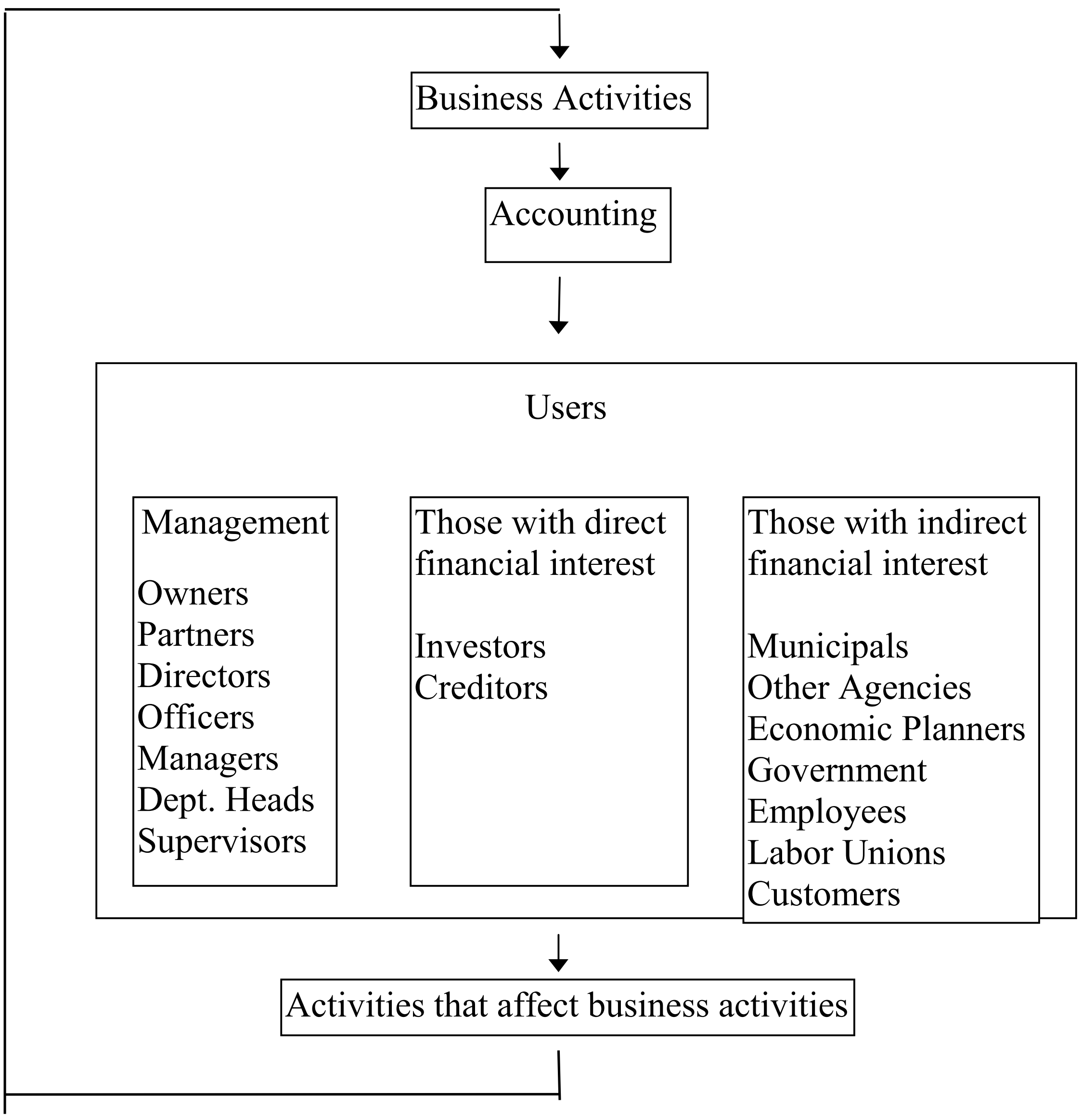


Figure 2.2 The users of an accounting system

Cash Receipts Journal, Cash Payment Journal and General Journal etc. in the accounting system of a company. Each journal can be considered as a subsystem of the accounting information system. Special purpose journals are designed to record particular kind of transactions. Thus, all transactions in a special purpose journal result in debits and credits to the same account. Each special purpose journal has relevant subsidiary ledger to post the transactions. Finally at the end of the accounting period, as an example, at the end of the month, total of the transactions will be posted to the main ledger called general ledger. In a computerized accounting system, these special purpose journals and subsidiary ledgers represent subsystems. All these subsystems interact with the main sub system called general ledger sub system which produces the final accounting information. General ledger sub system is straightforward, easy to develop and maintain. However, since the other subsystems interact with the different user groups, they are subject to frequent changes. For example, let us consider the payroll subsystem of a company. When there is a demand for salary increase from the employees, management has to increase salaries in order to fulfill the demand. This may introduce modifications to the payroll sub system. In this case, the general ledger sub system is not subject to modifications since it gets only few journal entries from the payroll subsystem. Figure 2.4 depicts this situation. Therefore subsystems of an accounting system should be flexible enough to accommodate the changes with short notice, sometimes within few hours. The Pilot System, which is going to be explained in the section 2.3, is also a subsystem of an accounting system of a travel agent.

2.2 Accounting language

Accounting is the language of business. It has well defined syntax and semantics. Therefore manual accounting system can be considered as an executable accounting specification.

2.2.1 Syntax

An accounting system consists of several accounts, journals, ledgers and reports. Information is kept in journals and ledgers in account units till they are used to generate management information.

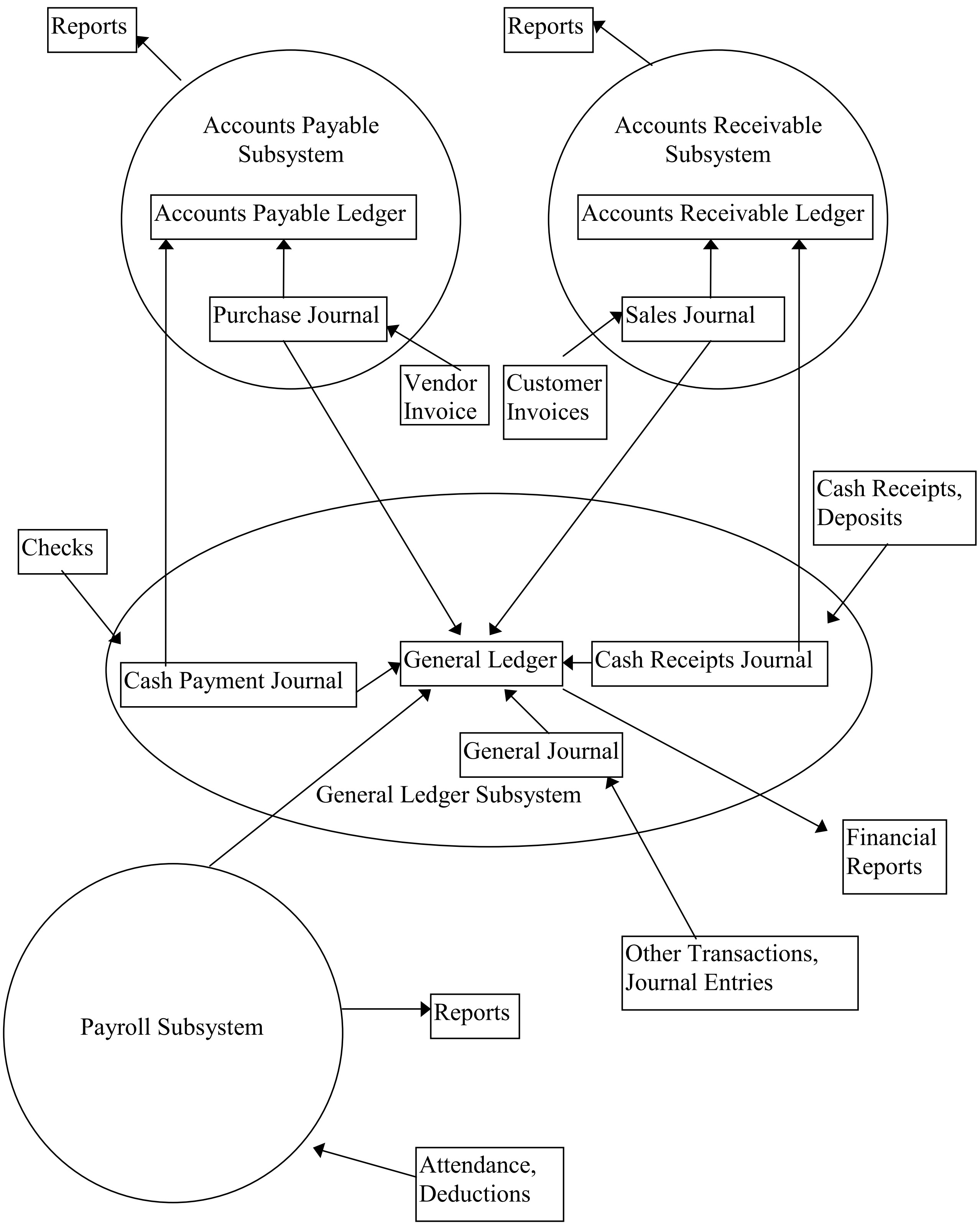


Figure 2.3 A typical accounting system

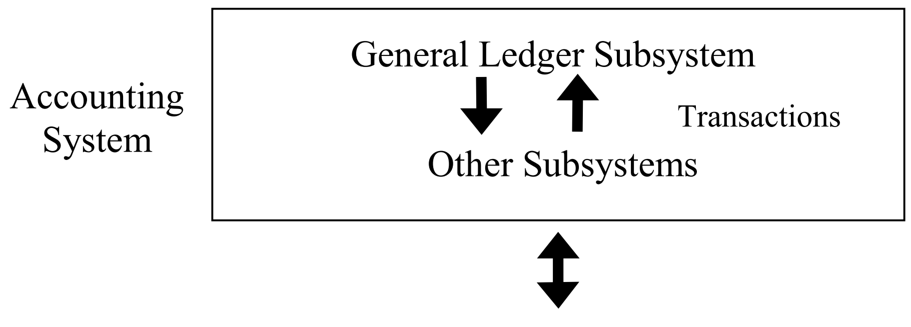


Figure 2.4 Interaction of accounting systems with environment

A basic accounting language consists of

1. Set of accounts.
2. Set of journals (At least General Journal).
3. Set of ledgers (At least General Ledger).
4. Set of reports.
5. A process called journalizing.
6. A process called posting.
7. A process called closing.
8. An accounting equation : Assets = Liability + Capital.
9. Business transactions such as receipts, payments etc.
10. Each account consists of at least name, date column, description column, debit column and credit column.
11. Accounts are grouped into three categories namely, Assets, Liabilities and Owner accounts.
12. Ledgers contain set of accounts.
13. Each transaction should contain date, description, amount.
14. Journals should have at least date column, account debited column, account credited column and amount column.

2.2.2 Semantics

Recording of information in various journals and ledgers are required to follow a well define procedure. These procedures give the necessary semantics for the accounting language.

Semantics of a basic accounting system are as follows:

1. Each transaction recorded in a journal should satisfy the accounting equation.
2. For each transactions, total debits should equal to total credits.
3. Each transaction should at least be credited to one account and debited to one account.
4. Increases in assets are recorded in debit side and decreases in the value of assets are recorded in credit side of an asset account.
5. Increases in all liabilities and owner's account are recorded as credits and decreases are recorded as debits.
6. Each transaction should be first recorded in a journal and then posted to a ledger.
7. All transactions should finally be posted to the general ledger.

2.3 The pilot system

This section explains the pilot system which is going to be used to explain the features of the specification language. The pilot system is a subsystem of an accounting system of a travel agent. This subsystem interacts with the passengers who request flight reservations and finally transactions are generated for the accounting system.

2.3.1 Transactions generation system of a travel agent

Let us consider a travel agent who provides air line reservation services to passengers. The transactions generation system has to create transactions to the accounting system after going through various phases of air line bookings. Requirements for the system are as follows:

1. The system is activated when a request for a flight reservation is received from a passenger.
2. When a request for a reservation is made, the consultant should record the passenger details such as name, destination, intended departure date, intended air line and maximum cost.
3. If passenger specifies a particular air line, then consultant should contact the requested air line and get the reservation details such as possible departure date, flight number, departure time, destination, reservation status and cost. Reservation status will be one of the followings: "Confirmed", "Waiting" or "Cancel".
4. If the passenger does not specify an air line then the consultant has to search for a suitable air line and get the reservation details.
5. When the consultant gets the reservation details from the air line, he should contact the passenger and inform.
6. If the passenger agrees with the reservation details then consultant has to reserve the flight and generate a reservation transaction to the accounting system.
7. If the passenger is not satisfied about the reservation then cancel the reservation and ask him whether the consultant should try again.
8. If the passenger wants to try again, then the consultant has to follow the instructions starting either from 3 or 4. If not, remove the passenger details from the records.
9. If the reservation status is waiting, then ask the passenger whether he is ready to wait or not. If he is ready to wait then wait for confirmation. If not, follow the instruction in 7.
10. If the reservation status is cancel follow the instructions in 7.

11. A passenger can cancel his request for reservation while he is waiting for confirmation without penalty.
12. Payment should be made after the reservation is confirmed and before departure in order to issue the ticket.
13. A passenger can cancel a reservation before payment.
14. If a passenger is made a cancel request after payment and 60 to 31 days before departure then cancel the reservation and charge him 10% of the cost. Generate the cancellation transaction to accounting system.
15. If a passenger is made a cancel request after payment and 30 to 15 days before departure then cancel the reservation and charge him 20% of the cost. Generate the cancellation transaction to accounting system.
16. If a passenger is made a cancel request after payment and 14 to 3 days before departure then cancel the reservation and charge him 30% of the cost. Generate the cancellation transaction to accounting system.
17. If a passenger is made a cancel request after payment and 2 days before departure then cancel the reservation and charge him 50% of the cost. Generate the cancellation transaction to accounting system.
18. If a passenger is made a cancel request after payment and on the day of departure then cancel the reservation and charge him 100% of the cost. Generate the cancellation transaction to accounting system.

2.4 Information systems' requirements

Requirements of an information systems consist of three major components.

1. What.
2. Who.
3. When.

These three components can be seen in any given information system. Therefore the specification language used to represent requirements should be capable of representing these three components. The component "What" considers the what should be done by a computerized system instead of how it can be done.

This is normally the main topic of the requirement engineering. Even though the component "Who" is not essential for a functioning of a computerized information system, it is more useful to know about responsible person for an action in maintenance phase, specially when end-users and system analysts are new to the computerized system. However, it is necessary to know about the "When" component. Real world information systems consist of series of actions which are chronologically ordered. Activation of a particular action is entirely depended on actions already activated. Therefore without specifying temporal relations between event and activities in an information systems concisely, precisely and accurately, it is impossible to build a computerized information system. Therefore this research pays special attention on representing incomplete relative information.

2.5 View of a system

Building of a computer system requires a well design model for the application. Since accounting is the main application area of this research, the model selected should be flexible enough to represent any accounting system. As described above, accounting systems have lots of chronologically ordered external and internal events and activities. An event invoke one or more actions in accounting systems. That is the temporal order of the events and activities are very important. These actions may trigger another set of actions. On the other way, generation of one information in accounting system will lead to generation of several other information. The selected model should be capable to represent these characteristics of accounting systems.

Entity-relationship models are widely used in data modeling than other models. However, entity-relationship modeling does not support the idea of creating new objects using one or more objects which is very important in accounting systems. For example, a particular journal can be considered as a collection of several similar transactions. In entity-relationship modeling, journals cannot be considered as a collection of transactions instead considered as two separate objects having some relation. Entity-relationship view of a system and another view of a system which can solve the problems with entity-relationship approach will be discussed in next sections.

2.5.1 Entity-relationship view of a system

In entity-relationship modeling approach, it is impossible to create a new entity with new attributes and attributes from parent entities as a result of a relationship between entities. It simply attaches entities to relationships. The new attributes are linked to the relationships and not to the entities [Chen 1976][Ullman 1982]. Bekke [Bekke 1992] shows that the entity-relationship modeling is not suitable for some complex problems. Figures 2.5 exhibits the entity-relationship representation of a relationship between Passengers and Flights, respectively.

2.5.2 View of a system adopted in this research

A system consists of objects which have properties as their attributes. Objects have relationships among them. These relationships will result in:

1. Modification to at least one attribute of the object, and/or
2. Creation of a new object with new attributes in addition to the original attributes.

New objects created will have relationships with other objects. However, Stefan Sigfried's idea is little different [Sigfried 1996] in declining with above phenomena. He suggests complex classes contain object structures in addition to attributes and methods. His suggestion is not adopted in this research, because the idea of using objects as attributes in another object is supported by the temporal logic. This will be discussed in the section 5.1 of this thesis. Therefore, whatever the knowledge representation scheme adopted, it should be capable of representing 1) modification to at least one attribute of the object and 2) creation of a new object with new attributes in addition to the original attributes. However, the second requirement cannot be fulfilled if the well known entity-relationship representation (Figure 2.5) is used. Therefore, view of a system shown in Figure 2.6 which can satisfy both requirements is adopted in this research. The system starts functioning when an external or an internal event triggers an activity/activities under specific conditions. These activities will trigger other activities and so on [Martin 1993]. Events, activities, rules and conditions consist of objects and relationships. There are two types of events; namely external events and internal events. External events occur due to environmental changes while internal events occur due to execution of programs.

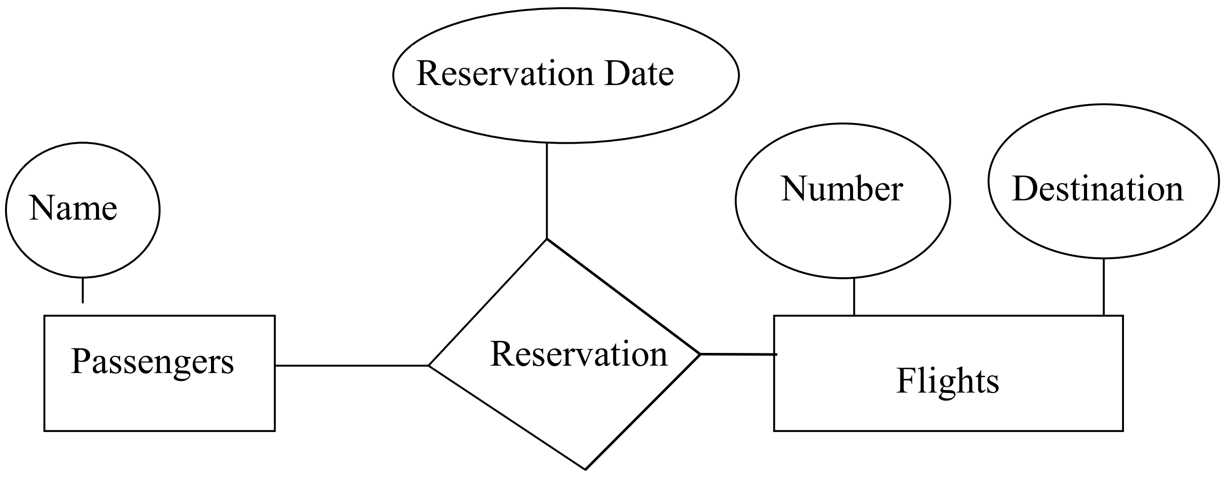


Figure 2.5 Entity-relationship representation

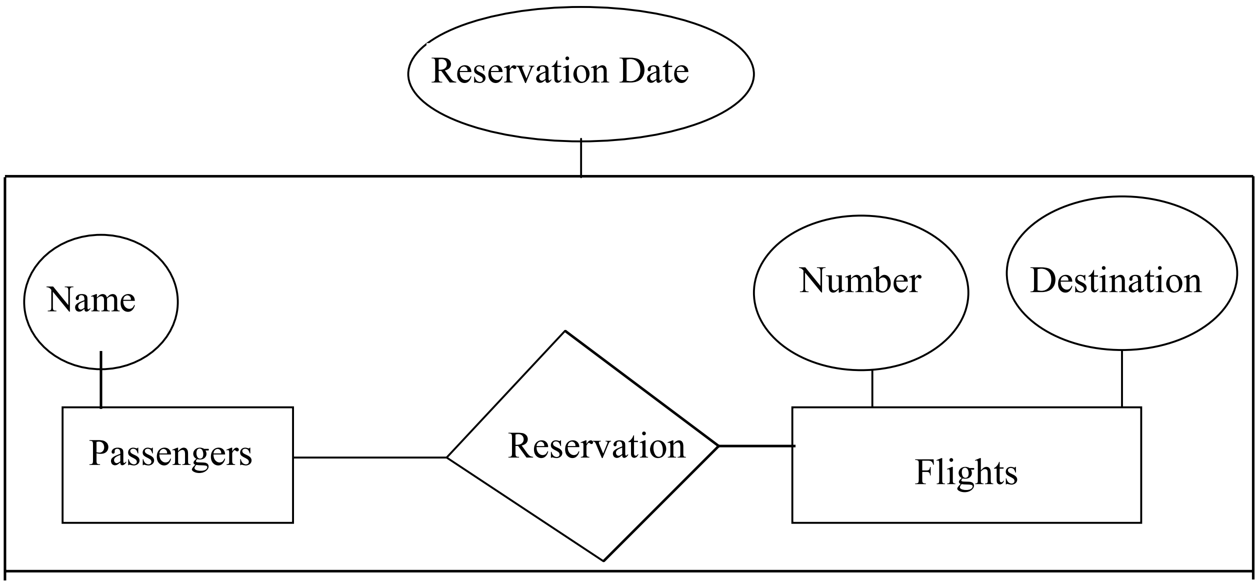


Figure 2.6 Representation adopted in this research

CHAPTER III

REPRESENTING TEMPORAL KNOWLEDGE

This chapter discusses importance of incomplete relative temporal knowledge in information systems, the present approaches to represent temporal knowledge; Allen’s method, McDermott’s method, Vilain and Kautz’s method, Temporal operators, Time-sensitive Boolean operators in detail. Then the problems of present approaches to temporal knowledge representation (problems inherited with point based and interval time models and the necessity of complete time information of occurrence of events to represent temporal knowledge) will be discussed. Finally it introduces the new logical connective TAND which can solve the problems of above representation methods (because it is not based on any time models) and shows how TAND can be used to represent Allen’s temporal relations and existing temporal operators.

3. 1 An incomplete temporal knowledge about events

While analyzing requirements of an information system, system analyst has to deal with what has already happened, what is happening now, what will happen and has to take some sort of actions according to them. However complete information about time of occurrence of an event may not be available. For example, in the transactions generating system, it is understood that a passenger will make a call for flight reservation but the exact time when he is going to call us is not available. For another example, there will be a cancellation after payments but the exact time when it will happen is not available. Therefore at the requirement analysis phase system analyst has to deal with the incomplete temporal knowledge about events rather than events with the complete information about exact time of occurrence. But the complete information about events may be required to answer a database query. However, to decide whether a passenger will make a cancellation request after the payment, it is not necessary to know the exact date of payment but it is enough to know that the payment is already made. Therefore it is sufficient to know that the passenger's payment is found in the payment database. That means decisions can be made with relative time information. In software requirements specifications the requirements are stated with relative time information since the exact time information are not available at this stage. These show the importance of the incomplete relative temporal information about events in software requirements analysis phase. Therefore software requirements specification languages should be capable of representing incomplete relative temporal information about events.

3.2 Conventional methods to represent temporal knowledge

Temporal knowledge representation models introduced so far can be grouped into two: namely point based and interval based temporal models. Point based time models represent time as a point in a real line while interval models represent time by time intervals. In point based models, time interval is represented as a set of time points which are bounded by start and end points. The following section describes conventional approaches to represent temporal knowledge.

3.2.1 Allen's method

Allen [Allen 1983] has developed a temporal logic in which time intervals are the primitives. He points out the weaknesses of point based temporal models. Allen shows that there can be a time point in point based model where either two events could occur at the same time or non could occur. For example, consider the two events "Switch-On" and "Switch-Off". Then according to Allen, there can be a time point where both "switch-On" and "Switch-Off" occur or none of them could occur. To solve this problem, he suggests time interval as the primitive for temporal representations. Time intervals can meet each other. Therefore "Switch-On" can meet "Switch-Off". Further he suggests thirteen possible relations between time intervals, namely "before", "after", "during", "contains", "overlaps", "overlapped-by", "meets", "met-by", "starts", "started-by", "finishes", "finished-by" and "equals". Figure 3.1 depicts them graphically. Allen uses these temporal relations to define events and activities. In his temporal logic, he applies first-order predicates to represent temporal relations [Allen 1984].

START(t1, t2)	: time interval t1 shares the same beginning as t2, but ends before t2 ends;
FINISHES(t1, t2)	: time interval t1 shares the same end as t2, but begins after t2 begins;
BEFORE(t1, t2)	: time interval t1 comes before interval t2, and they do not overlap in any way;
OVERLAP(t1, t2)	: interval t1 starts before t2, and they overlap;
MEETS(t1, t2)	: interval t1 comes before interval t2, but there is no any interval between them, i.e. t1 ends where t2 starts;
EQUAL(t1, t2)	: t1 and t2 are the same interval.

To represent property holding over an interval T, Allen needs two other predicates; HOLDS(p, T) and IN(t, T). HOLDS(p ,T) means that the property p holds over an interval T and IN(t, T) means that the subinterval t is in interval T. He defines IN as follows:

$\text{IN}(t1, t2) \Leftrightarrow (\text{DURING}(t1, t2) \vee \text{STARTS}(t1, t2) \vee \text{FINISHES}(t1, t2))$.

His definition for HOLDS(p, T) is

$\text{HOLDS}(p, T) \Leftrightarrow (\forall t.\ \text{IN}(t, T) \Rightarrow \text{HOLDS}(p, t))$.

The negation of a property p is defined as

$\text{HOLDS}(\text{not}(p), T) \Leftrightarrow (\forall t.\ \text{IN}(t, T) \Rightarrow \neg\text{HOLDS}(p, t))$.

That means if a property p holds over an interval T, it holds over all subintervals of T. However, Allen could not define events and actions using HOLDS predicate. Therefore he defines another predicate OCCUR. The reason for use of another predicate is due to his definition of HOLDS predicate. HOLDS(p , T) says that property p holds over an interval T as well as all subinterval of T. OCCUR(e, t) says that the event e, occurs in the time interval t. Therefore to represent “event e1 occurs before event e2”. It is necessary to write

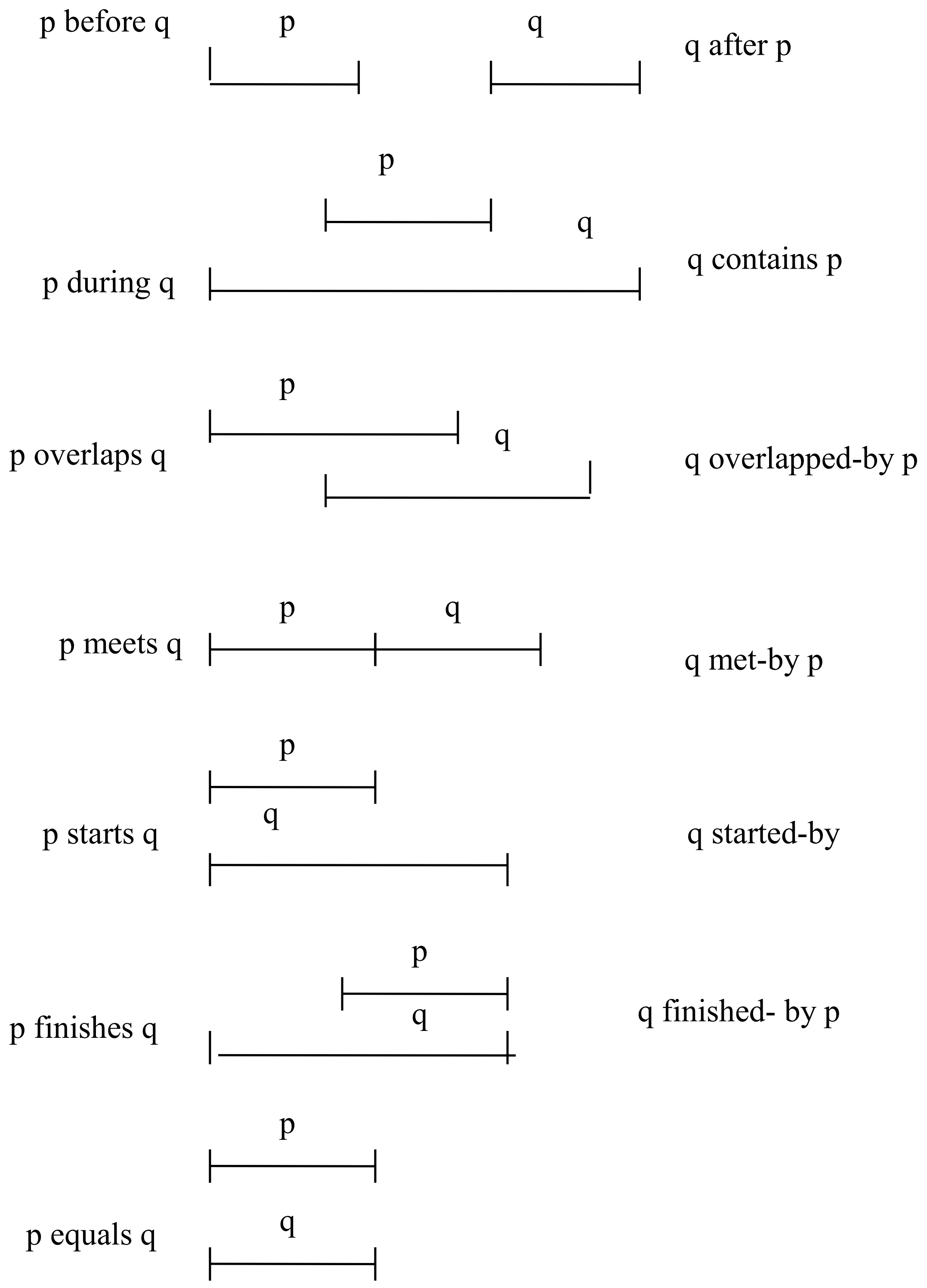


Figure 3.1 Allen's temporal relations

OCCUR(e1, t1) $\wedge$ OCCUR(e2, t2) $\wedge$ BEFORE(t1, t2).

However, Galton [Galton 1990] shows the weaknesses of Allen's temporal interval in representing continues changes. According to Allen's definition of HOLDS, to HOLDS(p, T) to be true p must be held in all subintervals of T. If p is continuously moving with time, p cannot be held in all subinterval at the same time. Hence if HOLDS(p, T) is true then p cannot move but has to rest over time interval T. Galton argues that in order to overcome this problem one must be prepared to treat time points and time intervals on an equal footing. Therefore we can think of three kind of time models:

1. Point based models,
2. Interval based models, and
3. Point-Interval based models.

3.2.2 McDermott's method

There are two key concepts in McDermott's temporal logic[McDermott 1982].

1. More than one event can occur starting at a given instant.
2. Many events do not occur discontinuously.

To capture these ideas, he defines universe as a collection of states. A state is an instantaneous snapshot of the universe. States are partially ordered by a relation "= <". To represent "s1 comes before or identical to s2", he writes (= < s1 s2). In this method, facts type and event types are used as building blocks. State (or time point) is used as the primitive. Mcdermott applies "Cambridge Polish" notation for logical formulas. Every term, atomic formula, and combination is of the form (p…), where p is a function, predicate or connective. His representation of universal quantifier, existential quantifier, "not" operator, "if" statement, "and" operator, "or" operator, "if and only if" statement and occurrence of events are as follows:

Universal quantifier (forall)	:	(forall (-variables-) formula)

Existential quantifier (exists)	:	(exists (-variables-) formula)
“not” operator	:	(not formula)
“if” statement	:	(if formula1 formula2)
“and” operator	:	(formula1 formula2…)
“or” operator	:	(or formula1 formula2…)
“if and only if” statement (iff)	:	(iff formula1 formula2)
Occurrences of events (Occ)	:	(Occ state1 state2 event)
“a” is an element of “x”	:	(elt a x).

Occ(state1 state2 event) says that the event “event” happens between states “state1” and “state2”. Unbound variables are prefixed with “?”. Therefore, “?s1” represents the unbound variable “s1”. The expression (< s1 s s2) represents that the state “s” is in between states “s1” and “s2”. A CHRONICLE is a complete possible history of the universe, a totally ordered set of states extending infinitely in time.

Definition of CHRONICLE :

```
(iff (is CHRONICLE ?x) ; x is a CHRONICLE
        (and ; a set of states
                (forall (y) (if (elt y ?x) (is STATE y)))
                ; for all y, if y is an element of  x then y is a STATE
                ; totally ordered
                (forall (s1 s2) ; for all s1, s2
                        (iff (and (elt s1 ?x) (elt s2 ?x))
                        ; if and only if s1 and s2 are elements of x then
                                (or (< s1 s2) (> s1 s2) (= s1 s2))))
                                ; state s1 comes before state s2, state s2 comes
                                ; before state s1 or state s1 and state s2 are
                                ; identical
                ; infinite time
                (forall (t) ; for al t
                        (exists (s) ; there exists s such that
                                (and (elt s ?x) (= (d s) t))))))
                                ; state s is an element of x and date of state s is t
```

where function “d” in (d s) gives the date of state “s”. Comments begin at “;”. Example : DAY is always followed by NIGHT and NIGHT by DAY. DAY and NIGHT never overlap.

DAY and NIGHT are mutually exclusive (except at boundaries):

```
    (if (and (Occ ?s1 ?s2 DAY) (Occ ?s3 ?s4 NIGHT))
    ; if DAY occurs between unbound states s1 and s2 and
    ; NIGHT occurs between unbound states s3 and s4
        (forall (s) ; for all states s
            (if (and (= < ?s1 s ?s2) ( = < ?s3 s ?s4))
            ; if state s is identical to unbound state s1 or s2 or
            ; in between unbound states s1 and s2 and
            ; state s is identical to unbound state s3 or s4 or
            ; in between unbound states s3 and s4 then
              (or (= s ?s2 ?s3)
              ; state s is identical to unbound state s2 or s3 or
               (= s ?s1 ?s4)))) )
              ; state s is identical to unbound state s1 or s4.
```

Each takes a nonzero amount of time :

```
    (if (or (Occ ?s1 ?s2 DAY) (Occ ?s1 ?s2 NIGHT)
    ; if DAY or NIGHT occurs between unbound states s1 and s2 then
      (< ?s1 ?s2)) ; unbound state s1 comes before unbound state s2.
```

Each follow the other:

```
    (follows DAY NIGHT) ; DAY follows NIGHT
    (follows NIGHT DAY) ; NIGHT follows DAY
    where
    (iff (follows ?e1 ?e2)
    ; If and only if unbound event e1 follows unbound event e2 then
      (if (Occ ?s1 ?s2 ?e1)
      ;  unbound event e1 occurs between unbound states s1 and s2
        (forall (ch) (if (elt ?s2 ch)
        ; for all CHRONICLE ch, if state s2 is an element of ch
                        (exists (s3) ; there exists s3 such that
                         (and elt s2 ch) ; state s2 is an element of ch and
                           (Occ ?s2 s3 ?e2)))))))
                           ; unbound event e2 occurs between
                           ; unbound states s2 and s3.
```

3.2.3 Vilain and Kautz's method

In Vilain and Kautz's point algebra or PA networks[Vilain 1986] [Peter 1992], there are three basic relations that can hold between two points: <, = and >. The relative information between two points is represented as a disjunction of the basic relations <, =, >. For example, the relation {<, =} between points A and B is represented by $(A < B) \lor (A = B)$. Start and end points of a time interval T are represented by T^- and T^+. For example, "time interval T1 before interval T2" can be expressed as follows:

T1 before T2 : $(T1^- < T1^+) \land (T2^- < T2^+) \land (T1^+ < T2^-)$.

However, all Allen's temporal relations cannot be represented using PA algebra. For example, "T1 meets T2" cannot be represented properly. Therefore the subset of Allen's temporal relations that can be represented using the relations $\{<, \leq, =, >, \geq, \neq\}$ are allowed in PA networks. Representation of Allen's thirteen interval relations using point algebra is shown in Figure 3.2.

3.2.4 Temporal operators

Temporal operators available in temporal logic [Kroger 1987][Barringer 1996][Manna 1995][Torsun 1995] are yet another approach to represent temporal information. However, application of these operators are limited to point based temporal models. Most computer science applications including program verifications [Manna 1995] employ these temporal operators. Standard temporal operators are as follows:

Next : *Next* A is true at time T if and only if A is true at time T+1.

Last : *Last* A is true at time T if and only if A is true at time T-1.

Always in the future : *Always in the future* A is true at time T if and only if A is true at all time point > T.

Always in the past : *Always in the past* A is true at time T if and only if A is true all time points < T.

Sometimes in the future : *Sometimes in the future* A is true at time T if and only if A is true at a time point T1 where T1 > T.

Sometimes in the past : *Sometimes in the past* A is true at time T if and only if A is true at a time point T1 where T1 < T.

Until : A *Until* B is true at time T if and only if B is true at a time point T1 and A is true at all the time points t where T < t < T1.

Since : A *Since* B is true at time T if and only if B is true at a time point T1 and A is true at all time points t where T1 < t < T.

However, these temporal operators are not expressible enough to represent Allen's temporal relations. Further they have weaknesses inherited from point based temporal models

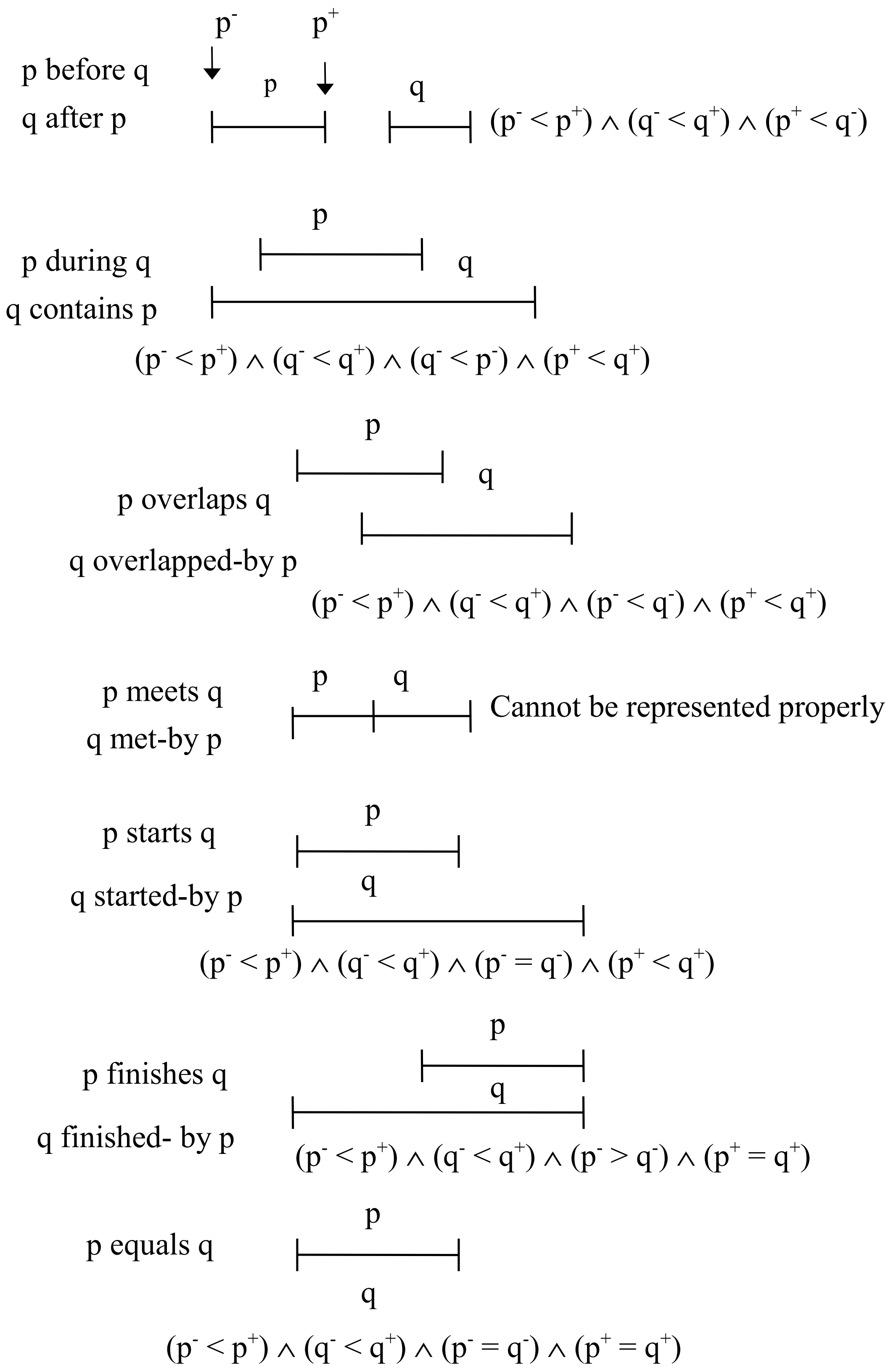

p^-
p^+
p before q
q after p
p
q
$(p^- < p^+) \wedge (q^- < q^+) \wedge (p^+ < q^-)$
p
p during q
q
q contains p
$(p^- < p^+) \wedge (q^- < q^+) \wedge (q^- < p^-) \wedge (p^+ < q^+)$
p
q
p overlaps q
q overlapped-by p
$(p^- < p^+) \wedge (q^- < q^+) \wedge (p^- < q^-) \wedge (p^+ < q^+)$
p meets q
p
q
Cannot be represented properly
q met-by p
p
p starts q
q
q started-by p
$(p^- < p^+) \wedge (q^- < q^+) \wedge (p^- = q^-) \wedge (p^+ < q^+)$
p
p finishes q
q
q finished- by p
$(p^- < p^+) \wedge (q^- < q^+) \wedge (p^- > q^-) \wedge (p^+ = q^+)$
p
p equals q
q
$(p^- < p^+) \wedge (q^- < q^+) \wedge (p^- = q^-) \wedge (p^+ = q^+)$

Figure 3.2 Allen's temporal relations using PA algebra

.

3.2.5 Time-sensitive Boolean operators

The time-sensitive Boolean operators suggested by Bassiouni et al. [Bassiouni 1994] are a different approach to represent and reason about temporal information. They argue that the standard Boolean connectives based on the True/ False logic are not suitable for handling events/activities defined over periods of time. For example, they say that the events p and q in (p AND q) are not necessary to occur at the same time. They define a new time-sensitive operator AND^S , OR^S and NOT^S to represent the concurrent occurrence of events. Hence the events p and q in (p AND^S q) occur simultaneously. Their definition of AND^S is as follows:

Definition:

If S1 and S2 are two sets of time intervals, then the expression S1 AND^S S2 returns the null set ϕ if S1 and S2 do not have any overlapping intervals. Otherwise AND^S returns the time sub-intervals that are contained in both S1 and S2 (i.e. it returns their overlapping).

Similarly, they define time-sensitive OR^S and NOT^S.

Definition:

The time-sensitive unary operator NOT^S returns the set of intervals which is the complement of its operand with respect to the universal interval <0, NOW>. Notice therefore that if S is a set of time intervals, then the expression NOT^S (S) returns ϕ only if the union of the intervals of S is the universal interval.

Definition:

If S1 and S2 are two sets of time intervals, then the expression S1 OR^S S2 returns ϕ only if both S1 and S2 are null sets. Otherwise, it returns the union of the intervals of S1 and S2.

Time intervals are defined over integer time points. Further they show that the time-sensitive Boolean operators satisfy the properties of standard Boolean connectives.

3.2.6 Problems with the above methods

All temporal knowledge representation methods discussed above except temporal operators, require complete time information to represent the temporal information about events. They need exact time of occurrence of events. However, as described in the section 3.1, information systems consist of incomplete relative temporal knowledge rather than having complete time information of occurrence of events. Therefore these methods can not be used in software specifications to represent incomplete relative temporal knowledge. The temporal operators available in the literature are not expressible enough to represent temporal relations suggested by J. F. Allen [Allen 1983]. On the other hand, temporal operators inherit the weaknesses of point based models. Temporal operators are more suitable in program verifications rather than software requirement specifications. Although Allen's temporal relations require complete time information of events, software requirements specifications require them only to represent the temporal information. However, Allen's temporal intervals have weaknesses in representing continues changes. Therefore a temporal knowledge representation method which can represent Allen's relative temporal relations with incomplete time information of occurrence of events is required. This thesis suggests a new method to represent incomplete relative temporal knowledge [Wijayarathna 1997] which can overcome these problems.

3.3 The AND_THEN (TAND) connective

3.3.1 The standard logical connective AND

Logical connective AND ($\wedge$) does not discriminate the temporal order of events. Even though two events occur in different time zones, as far as the standard logical connective AND is concerned, they occur in the same time zone. Hence it treats the events as concurrent events. The whole time line is a single time zone for the AND connective. The "time zone" represents both time points and time intervals. Therefore the standard logical connective AND is not suitable for applications where the temporal order of events play an important role. As describe in the previous section, Bassiouni et al.[Bassiouni 1994] argues that events p and q in "$p \wedge q$" are not necessary to occur simultaneously. However, "$p \wedge \neg p = \text{false}$" is one of the universally accepted properties of the standard AND connective (Idempotent law). Whenever p is true, ¬p becomes false and when p is false, ¬p becomes true. Therefore in both cases "($p \wedge \neg p$) = (true $\wedge$ false)= (false $\wedge$ true)" is false. According to their argument p and ¬p are not required to occur simultaneously; i.e. it is not required for "$p \wedge \neg p$" to be false. Because p can be true at one time and ¬p can be true at another time giving " true $\wedge$ true = true". This contradicts with the universally accepted property "$p \wedge \neg p = \text{false}$". However the standard logical connective AND considers events p and q in "$p \wedge q$" as having occurred simultaneously, even though they really occur in two different time zones. Hence it is more realistic and logical to restrict standard logical connective AND to represent simultaneous events and to introduce the new temporal logical connective to represent other cases.

3.3.2 The TAND ($\prod$) logical connective

For any given two events p and q, the truth state at any time “t” will be one of the following:

1. Only p occurs,
2. Only q occurs,
3. Both p and q occur:
   3.1 Both p and q occur simultaneously,
   3.2 p and q occur sequentially,
4. Neither p nor q occur.

Figure 3.3 shows these situations graphically.

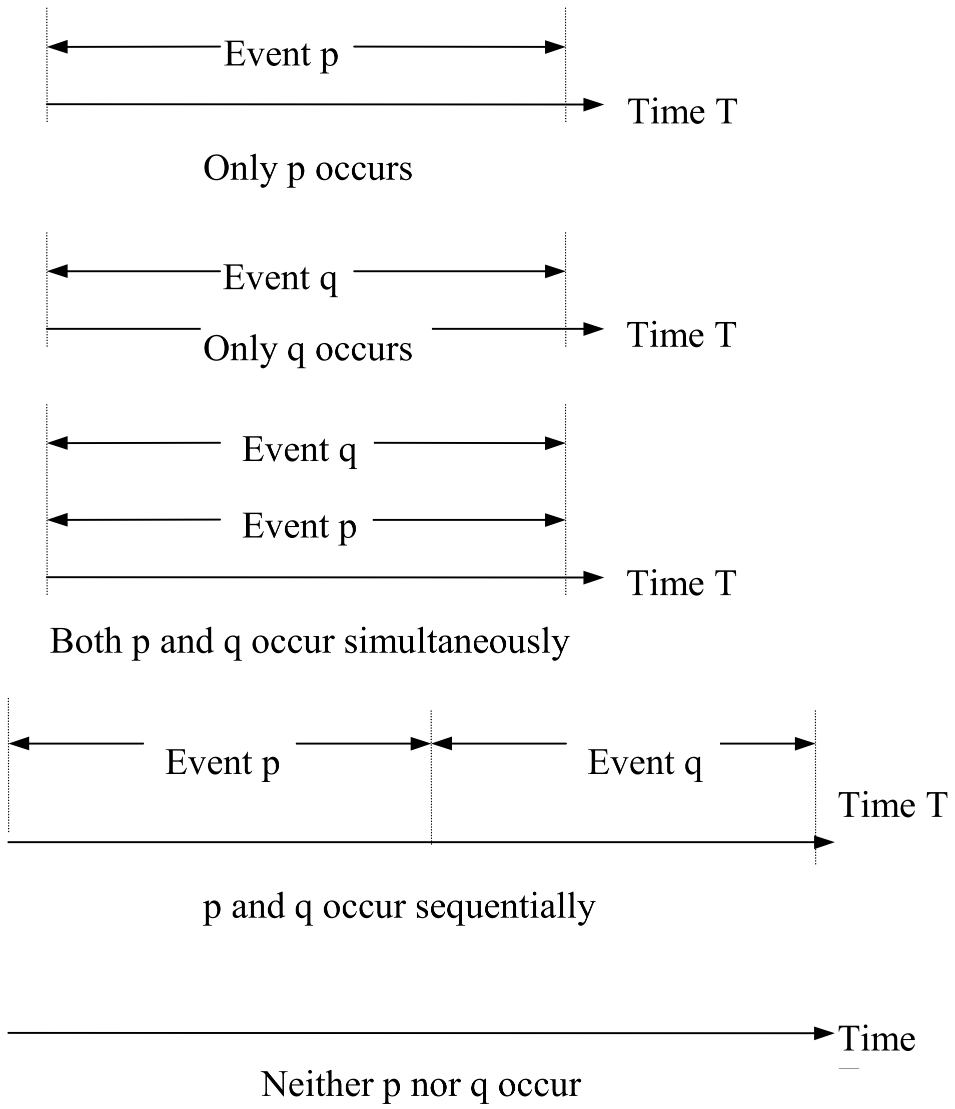


Figure 3.3 Possible occurrences of two events

That is at any given time t,

1. $p = \text{true}$,
2. $q = \text{true}$,

3.1 $(p \wedge q) = \text{true}$,

3.2 ??? (cannot represent using existing connectives) or

4. $(\neg p \wedge \neg q) = \text{true}$.

Here $\wedge$ represents concurrent events. Therefore we can represent any relative temporal relation using these four truth states provided that we have a method to represent the sequential occurrence of events. For example to represent "p meet q" , we have to express that there exists a state where "p = true " ***and then*** another state where "q = true ". That is "p meet q" is true, iff { p = true ***and then*** q = true} is true. In this representation one state is followed by another state. As discussed in the section 3.3.1, the standard logical connective AND does not distinguish whether p and q occur simultaneously or not. Hence what is required is a logical connective to represent ***and then*** in such a way that "true ***and then*** true" is equal to true. The ***and then*** is defined as TAND logical connective and is represented by $\prod$ in this thesis.

Definition:

If p and q are true in time zones T1 and T2 respectively. Then $(p \prod q)$ is true if and only if

a) q is true immediately after p is true.
b) Both p and q are not true at the same time.
c) At any given time, either p or q is true.

Axioms

Now three important axioms about TAND ($\prod$) can be defined.

A1. $(p \prod q) \Rightarrow ((p \vee q) \wedge \neg(p \wedge q))$

A2. $(p \prod q) \neq (q \prod p)$

A3. Time zone T2 follows time zone T1.

Therefore at any given time t, if (p $\prod$ q) is true then either p or q must be true. However both p and q cannot be true or false simultaneously.

Properties of AND and TAND

1. Idempotent Law

$p \wedge \neg p = \text{false}$; either p is true or false.
$p \prod \neg p = \text{false}$; p is true *and then* p is true.
p is false *and then* p is true.
p is false *and then* p is false.
$p \prod \neg p = \text{true}$; p is true *and then* p is false.

Therefore TAND does not obey the Idempotent law as AND does.

2. Commutative law

$p \wedge q = q \wedge p$
$p \prod q \neq q \prod p$

Since p $\prod$ q means that p is true *and then* q is true while q $\prod$ p means that q is true *and then* p is true. Two things can be ascertained from p $\prod$ q. First, p and q do not occur at the same time and second, if T1 and T2 are sets of time zones where q and p occur respectively then $\forall t1 \in T1 > \forall t2 \in T2$. Therefore TAND does not obey the commutative law as AND does. However both "p $\wedge$ q" and "q $\wedge$ p" express that p and q occur at the same time. Therefore the order of events is insignificant.

3. Distributive law

$p \wedge (q \vee s) = (p \wedge q) \vee (p \wedge s)$
$p \vee (q \wedge s) = (p \vee q) \wedge (p \vee s)$
$p \prod (q \vee s) = (p \prod q) \vee (p \prod s)$ (table 1)
$p \prod (q \wedge s) = (p \prod q) \wedge (p \prod s)$ (table 2)

TAND obeys the distributive law as AND does. The truth tables, Table 1 and Table 2 show this in detail.

4. Associative law

$$p \wedge (q \wedge s) = (p \wedge q) \wedge s$$
$$p \prod (q \prod s) = (p \prod q) \prod s$$

TAND obeys the associative law as AND does because associative law does not change the order of the occurrence of events.

5. DeMorgan's law

$$\neg(p \wedge q) = \neg p \vee \neg q$$
$$\neg(p \prod q) \neq \neg p \vee \neg q$$

If TAND obeys DeMorgan's law then $\neg(p \prod q) = \neg p \vee \neg q$. Since the conventional logical connective $\wedge$ obeys the DeMorgan's law, $\neg p \vee \neg q = \neg(p \wedge q)$. However, by the definition, $(p \prod q)$ should be false when $(p \wedge q)$ is true. Therefore, TAND does not obey DeMorgan's law.
As shown above TAND obeys distributive and associative laws as AND does but not other laws.

(q and s occur in time zone T2 and p occurs in time zone T1 ; T1 follows T2)

p	q	s	$(q \vee s)$	$(p \prod q)$	$(p \prod s)$	$p \prod (q \vee s)$	$(p \prod q) \vee (p \prod s)$
T	T	T	T	T	T	T	T
T	T	F	T	T	F	T	T
T	F	T	T	F	T	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	F	F	F	F	F

Table 1 Truth table for distributive law (OR)

p	q	s	$(q \wedge s)$	$(p \prod q)$	$(p \prod s)$	$p \prod (q \wedge s)$	$(p \prod q) \wedge (p \prod s)$
T	T	T	T	T	T	T	T
T	T	F	F	T	F	F	F
T	F	T	F	F	T	F	F
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	F	F	F	F	F
F	F	T	F	F	F	F	F
F	F	F	F	F	F	F	F

Table 2 Truth table for distributive law (AND)

Case 1.

Suppose (p TAND q) is true and p is true. Now for axiom A1 to be true, right hand side of axiom A1 must be true, since the left hand side is true. (A $\Rightarrow$ B is true iff B is true or A is false.) That is,

$(p \text{ TAND } q) \Rightarrow (q \wedge \neg p) \vee (p \wedge \neg q)$

$\Rightarrow (q \wedge \neg p) \vee (p \wedge \neg q) = T$ (since (p TAND q) = T)

$\Rightarrow (q \wedge F) \vee (T \wedge \neg q) = T$ (since (p = T))

$\Rightarrow F \vee \neg q = T$ (since $(q \wedge F) = F$ and $(T \wedge \neg q) = \neg q$)

$\Rightarrow \neg q = T$ (since $F \vee \neg q = \neg q$)

$\Rightarrow q = F$

Starting with both (p TAND q) and p are true, it is inferred that q is false. Therefore it proves that if (p TAND q) is true and p is true then q must be false.

Case 2.

Now let us assume that (p TAND q) is true and q is true. Then applying the same logic it can be shown that always p must be false.

Case 3.

Let us say that (p TAND q) is true and p is false.

Then, $(p \text{ TAND } q) \Rightarrow (q \wedge \neg p) \vee (p \wedge \neg q)$

$\Rightarrow (q \wedge \neg p) \vee (p \wedge \neg q) = T$ (since (p TAND q) = T)

$\Rightarrow$ (q $\wedge$ T) $\vee$ (F $\wedge$ $\neg$q) = T (since (p = F))

$\Rightarrow$ q $\vee$ F = T (since (q $\wedge$ T) = q and (F $\wedge$ $\neg$q) = F)

$\Rightarrow$ q = T (since q $\vee$ F = q)

This means whenever (p TAND q) is true and p is false, q must be true.

Case 4.

Now let us say that (p TAND q) is true and q is false. Then it can be shown that p must be true.

Case 5.

In contrast to the other cases, suppose that (p TAND q), p and q are true.

(p TAND q) $\Rightarrow$ (q $\wedge$ $\neg$p) $\vee$ (p $\wedge$ $\neg$q)

$\Rightarrow$ (q $\wedge$ $\neg$p) $\vee$ (p $\wedge$ $\neg$q) = T (since (p TAND q) = T)

$\Rightarrow$ (T $\wedge$ F) $\vee$ (T $\wedge$ F) = T (since (p = q = T))

$\Rightarrow$ F $\vee$ F = T

$\Rightarrow$ F = T

That is a contradiction. Therefore both p and q cannot be true at the same time when (p TAND q) is true.

Case 6.

Now let us assume that (p TAND q) is true and both p and q are false.

(p TAND q) $\Rightarrow$ (q $\wedge$ $\neg$p) $\vee$ (p $\wedge$ $\neg$q)

$\Rightarrow$ (q $\wedge$ $\neg$p) $\vee$ (p $\wedge$ $\neg$q) = T (since (p TAND q) = T)

$\Rightarrow$ (F $\wedge$ T) $\vee$ (F $\wedge$ T) = T (since (p = q = F))

$\Rightarrow$ F $\vee$ F = T

$\Rightarrow$ F = T

This shows that both p and q cannot be false at the same time when (p TAND q) is true.

Therefore considering above cases 1, 2, 3, 4, 5 and 6 it can be concluded that whenever (p TAND q) is true, either p or q should be true. However both p and q cannot be true or false simultaneously. This is a significant result as it proves that TAND can solve the problem that occur in point based temporal models. Allen[Allen 1983] says that there exists a time point in point based models where either ($p \wedge q$) is true or ($\neg p \wedge \neg q$) is true. This means that the truth state of any given time t in point based temporal models can be one of the following: only p is true, only q is true, ($p \wedge q$) is true or ($\neg p \wedge \neg q$) is true. However it is proved that there are only two truth states when (p TAND q) is true (only p is true or only q is true). This is Allen's[Allen 1983] (p meets q) temporal relation. Since TAND is not based on any time models, it can be used to represent point based applications, interval based application as well as point-interval based applications.

3.3.3 Representation of Allen's temporal relations using TAND

The important factor to remember when representing temporal relations with TAND is that the events p and q should be mutually exclusive. That is to say always (p AND q) should be false. Events p and q can contain more than one event. In this situation p and q can be considered as composite events. In this thesis events represent single events as well as composite events. Since it is already shown how to represent (p meets q) using $\prod$, in this section, representation of other temporal relations using TAND will be shown.

First let us consider the case of ($A \prod B \prod C$) where A, B, and C are three mutually exclusive events. By definition of TAND,

a) ($A \wedge B \wedge C$) should be false.
b) Time of occurrence of A< time of occurrence of B < time of occurrence of C.

The ($A \prod B \prod C$) is equal to ($A \prod B$) $\prod$ C because TAND obeys associative law. Therefore expressions having more than two mutually exclusive events can be decomposed into sub expressions having at most two events keeping the meaning of the expression unchanged.

Then by axiom A1,

$(A \prod B) \Rightarrow (A \vee B) \wedge \neg(A \wedge B)$.

If $(A \prod B)$ is D, then $(A \prod B \prod C) = (D \prod C)$.

Applying axiom A1 again we have

$(D \prod C) \Rightarrow (D \vee C) \wedge \neg (D \wedge C)$.

Then

$(A \prod B \prod C) = (D \prod C) \Rightarrow (D \vee C) \wedge \neg (D \wedge C) = (D \wedge \neg C) \vee (\neg D \wedge C)$

$\Rightarrow [((A \wedge \neg B) \vee (\neg A \wedge B)) \wedge \neg C] \vee [\neg((A \vee B) \wedge \neg(A \wedge B)) \wedge C]$

$\Rightarrow (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (B \wedge A \wedge C) \vee (\neg A \wedge \neg B \wedge C)$

Since A, B and C are mutually exclusive events, $(B \wedge A \wedge C)$ is false.

$\Rightarrow (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C)$ (F1)

$\Rightarrow$ (A=T, B=F, C=F) or (A=F, B=T, C=F) or (A=F, B=F, C=T)

$\Rightarrow$ Only one of A, B and C can occur at any given time t

Therefore it is enough to show that $(X \wedge Y \wedge Z)$ is false to prove the accuracy of any given temporal representation $(X \prod Y \prod Z)$.

p before q $= p \prod (\neg p \wedge \neg q) \prod q$

p before q is true, iff only p is true *and then* $(\neg p \wedge \neg q)$ is true *and then* only q is true. Since $(p \wedge \neg p \wedge \neg q \wedge q)$ is always false, at any given time t, only p can occur, only q can occur or both p and q cannot occur. In addition to this TAND gives the chronological order of the events. Therefore the representation of (p before q) is sound. If A= p, B = $(\neg p \wedge \neg q)$, C = q then applying F1

p before q

$\Rightarrow [p \wedge \neg(\neg p \wedge \neg q) \wedge \neg q] \vee [\neg p \wedge (\neg p \wedge \neg q) \wedge \neg q] \vee [\neg p \wedge \neg(\neg p \wedge \neg q) \wedge q]$

(p before q) $\Rightarrow [p \wedge \neg(\neg p \wedge \neg q) \wedge \neg q] \vee [\neg p \wedge \neg q] \vee [\neg p \wedge \neg(\neg p \wedge \neg q) \wedge q]$

If (p before q) = T and p = T then,

$(p \text{ before } q) \Rightarrow [T\wedge\neg(F\wedge\neg q)\wedge\neg q]\vee [F\wedge\neg q]\vee [F \wedge \neg(F \wedge \neg q) \wedge q] = T$

$\Rightarrow [T \wedge \neg F \wedge \neg q] \vee F \vee F] = T$

$\Rightarrow [T \wedge T \wedge \neg q] \vee F \vee F] = T$

$\Rightarrow [\neg q \vee F] = T$

$\Rightarrow \neg q = T$

$\Rightarrow q = F$

If (p before q) = T and p = F then,

$(p \text{ before } q) \Rightarrow [F\wedge \neg(T\wedge\neg q) \wedge\neg q]\vee [T\wedge\neg q]\vee [T \wedge \neg(T\wedge\neg q) \wedge q] = T$

$\Rightarrow [F\wedge \neg(\neg q) \wedge\neg q]\vee [T\wedge\neg q]\vee [T \wedge \neg(\neg q) \wedge q] = T$

$\Rightarrow [F\wedge q \wedge\neg q]\vee [\neg q]\vee [q \wedge q] = T$

$\Rightarrow [F \vee \neg q \vee (q \wedge q)] = T$

$\Rightarrow [\neg q \vee q] = T$

$\Rightarrow q = F \text{ or } q = T$

If (p before q) = T and q = T then,

$(p \text{ before } q) \Rightarrow[p \wedge\neg(\neg p\wedge F)\wedge F]\vee [\neg p \wedge F]\vee [\neg p \wedge\neg(\neg p \wedge F)\wedge T] = T$

$\Rightarrow[p \wedge\neg(F)\wedge F]\vee [\neg p \wedge F]\vee [\neg p \wedge\neg(F)\wedge T] = T$

$\Rightarrow[p \wedge T\wedge F]\vee [\neg p \wedge F]\vee [\neg p \wedge T\wedge T] = T$

$\Rightarrow [F \vee F \vee \neg p] = T$

$\Rightarrow \neg p = T$

$\Rightarrow p = F$

If (p before q) = T and q = F then,

$(p \text{ before } q)\Rightarrow [p \wedge\neg(\neg p \wedge T) \wedge T]\vee [\neg p \wedge T]\vee [\neg p \wedge\neg(\neg p \wedge T)\wedge F] = T$

$\Rightarrow [p \wedge \neg(\neg p \wedge T) \wedge T] \vee [\neg p \wedge T] \vee [\neg p \wedge \neg(\neg p \wedge T) \wedge F] = T$

$\Rightarrow [p \wedge \neg(\neg p) \wedge T] \vee [\neg p \wedge T] \vee [\neg p \wedge \neg(\neg p) \wedge F] = T$

$\Rightarrow [p \wedge p \wedge T] \vee [\neg p \wedge T] \vee [\neg p \wedge p \wedge F] = T$

$\Rightarrow [\ p\ ]\ \vee [\neg p]\ \vee [F\ ] = T$
$\Rightarrow p = T$ or $p = F$

If (p before q) = T and p = q = T, then
(p before q)
$\Rightarrow [\ p \wedge \neg(\neg p\wedge\neg q)\wedge\neg q\ ] \vee [\neg p\wedge\neg q\ ] \vee [\neg p\wedge\neg(\neg p\wedge\neg q) \wedge q\ ] = T$
$\Rightarrow [\ T \wedge \neg(F\wedge F)\wedge F\ ] \vee [F\wedge F\ ] \vee [F\wedge\neg(F\wedge F) \wedge F\ ] = T$
$\Rightarrow [\ F \vee F \vee F\ ] = T$
$\Rightarrow F\ = T$

This means that both p and q cannot be true at the same time when (p before q) is true.

If (p before q) = T and p = q = F, then
(p before q)
$\Rightarrow [\ p \wedge \neg(\neg p\wedge\neg q)\wedge\neg q\ ] \vee [\neg p\wedge\neg q\ ] \vee [\neg p\wedge\neg(\neg p\wedge\neg q) \wedge q\ ] = T$
$\Rightarrow [\ F \wedge \neg(T\wedge T)\wedge T\ ] \vee [T\wedge T\ ] \vee [T\wedge\neg(T\wedge T) \wedge F\ ] = T$
$\Rightarrow [\ F \vee T \vee F\ ] = T$
$\Rightarrow T = T$

This means that both p and q can be false at the same time when (p before q) is true.

This shows that at any given time t, only p occurs or only q occurs but both cannot occur. That is the combination of events required to occur (p before q) or (q after p). TAND connects them so as to give the required chronological order.

p overlaps q $=$ $(\ p \wedge \neg q)\ \Pi\ (\ p \wedge q\ )\ \Pi\ (\ q \wedge \neg p\ )$

p overlaps q is true, iff p is true *and the**n*** (p $\wedge$ q) is true *and the**n*** q is true. If the expression "overlaps" is accurate, it would be able to show that for any given time t, only p occurs, only q occurs or both p and q occur. Since $[\ (\ p \wedge \neg q) \wedge (\ p \wedge q\ ) \wedge (\ q \wedge \neg p\ )\ ]$ is false, the representation of (p overlaps q) is accurate.

Let us say $A = (p \wedge \neg q)$, $B = (p \wedge q)$ and $C = (q \wedge \neg p)$. Then from F1,
(p overlaps q) = $(A \prod B \prod C) \Rightarrow (A\wedge\neg B\wedge\neg C)\vee(\neg A\wedge B\wedge\neg C)\vee(\neg A\wedge\neg B\wedge C)$
$\Rightarrow((p\wedge\neg q)\wedge\neg(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge\neg(p\wedge q)\wedge(q\wedge\neg p))$.

If (p overlaps q) = T and p = T, then

(p overlaps q) = T
$\Rightarrow((p\wedge\neg q)\wedge\neg(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge\neg(p\wedge q)\wedge(q\wedge\neg p)) = T$
$\Rightarrow((T\wedge\neg q)\wedge\neg(T\wedge q)\wedge\neg(q\wedge F))\vee(\neg(T\wedge\neg q)\wedge(T\wedge q)\wedge\neg(q\wedge F))\vee(\neg(T\wedge\neg q)\wedge\neg(T\wedge q)\wedge(q\wedge F)) = T$
$\Rightarrow((\neg q)\wedge\neg(q)\wedge\neg(F))\vee(\neg(\neg q)\wedge(q)\wedge\neg(F))\vee(\neg(\neg q)\wedge\neg(q)\wedge(F)) = T$
$\Rightarrow(\neg q\wedge\neg q\wedge T)\vee(q\wedge q\wedge T)\vee(q\wedge\neg q\wedge F) = T$
$\Rightarrow(\neg q\vee q\vee F) = T$
$\Rightarrow q = T$ or $q = F$

If (p overlaps q) = T and p = F, then

(p overlaps q) = T
$\Rightarrow((p\wedge\neg q)\wedge\neg(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge\neg(p\wedge q)\wedge(q\wedge\neg p)) = T$
$\Rightarrow((F\wedge\neg q)\wedge\neg(F\wedge q)\wedge\neg(q\wedge T))\vee(\neg(F\wedge\neg q)\wedge(F\wedge q)\wedge\neg(q\wedge T))\vee(\neg(F\wedge\neg q)\wedge\neg(F\wedge q)\wedge(q\wedge T)) = T$
$\Rightarrow(F\wedge\neg(F)\wedge\neg(q))\vee(\neg(F)\wedge(F)\wedge\neg(q))\vee(\neg(F)\wedge\neg(F)\wedge(q)) = T$
$\Rightarrow(F\vee F\vee(T\wedge T\wedge q) = T$
$\Rightarrow(F\vee F\vee q) = T \Rightarrow q = T$
If (p overlaps q) = T and q = T, then

(p overlaps q) = T
$\Rightarrow((p\wedge F)\wedge\neg(p\wedge T)\wedge\neg(T\wedge\neg p))\vee(\neg(p\wedge F)\wedge(p\wedge T)\wedge\neg(T\wedge\neg p))\vee(\neg(p\wedge F)\wedge\neg(p\wedge T)\wedge(T\wedge\neg p)) = T$

$\Rightarrow(F\wedge\neg p\wedge p) \vee (\neg F\wedge p\wedge p) \vee (\neg F\wedge \neg p\wedge \neg p) = T$

$\Rightarrow(F \vee p \vee \neg p) = T$

$\Rightarrow$ p = T or p = F

If (p overlaps q) = T and q = F, then

(p overlaps q) = T

$\Rightarrow((p\wedge T)\wedge\neg(p\wedge F)\wedge\neg(F\wedge\neg p))\vee(\neg(p\wedge T)\wedge(p\wedge F)\wedge\neg(F\wedge\neg p)) \vee (\neg(p\wedge T)\wedge \neg(p\wedge F)\wedge (F\wedge\neg p)) = T$

$\Rightarrow((p)\wedge\neg(F)\wedge\neg(F))\vee(\neg(p)\wedge(F)\wedge\neg(F)) \vee (\neg(p)\wedge \neg(F)\wedge (F)) = T$

$\Rightarrow(p\wedge T\wedge T) \vee (\neg p\wedge F\wedge T) \vee (\neg p\wedge T\wedge F) = T$

$\Rightarrow(p \vee F \vee F) = T$

$\Rightarrow p = T$

If (p overlaps q) = T and p = q = T, then

(p overlaps q) = T

$\Rightarrow ((p\wedge\neg q)\wedge\neg(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge(p\wedge q)\wedge\neg(q\wedge\neg p)) \vee (\neg(p\wedge\neg q)\wedge \neg(p\wedge q)\wedge (q\wedge\neg p)) = T$

$\Rightarrow((T\wedge F)\wedge\neg(T\wedge T)\wedge\neg(T\wedge F))\vee(\neg(T\wedge F)\wedge(T\wedge T)\wedge\neg(T\wedge F))\vee(\neg(T\wedge F)\wedge \neg(T\wedge T)\wedge (T\wedge F)) = T$

$\Rightarrow(F\wedge F\wedge T)\vee(T\wedge T\wedge T)\vee(T\wedge F\wedge F) = T$

$\Rightarrow F\vee T \vee F = T$

$\Rightarrow T = T$

That is, both p and q can be true at the same time when (p overlaps q) is true.

If (p overlaps q) = T and p = q = F, then

(p overlaps q) = T

$\Rightarrow ((p\wedge\neg q)\wedge\neg(p\wedge q)\wedge\neg(q\wedge\neg p))\vee(\neg(p\wedge\neg q)\wedge(p\wedge q)\wedge\neg(q\wedge\neg p)) \vee (\neg(p\wedge\neg q)\wedge \neg(p\wedge q)\wedge (q\wedge\neg p)) = T$

$\Rightarrow((F\wedge T)\wedge\neg(F\wedge F)\wedge\neg(F\wedge T))\vee(\neg(F\wedge T)\wedge(F\wedge F)\wedge\neg(F\wedge T))\vee(\neg(F\wedge T)\wedge \neg(F\wedge F)\wedge (F\wedge T)) =T$

$\Rightarrow(F\wedge T\wedge T)\vee(T\wedge F\wedge T)\vee(T\wedge T\wedge F) =T$

$\Rightarrow F\vee F \vee F =T$

$\Rightarrow F =T$

That is, both p and q cannot be true at the same time when (p overlaps q) is true.

This indicates that at any given time t only p occurs, both p and q occur or only q occurs. That is the situation where (p overlaps q) or (q overlapped-by p) occur. Proper chronological order represented by TAND connective gives the required representations for (p overlaps q) or (q overlapped-by p).

p starts q $= (p \wedge q) \prod (q \wedge \neg p)$

(p starts q) is true, iff $(p \wedge q)$ is true and then q is true. Hence at any time, either $(p \wedge q)$ or $(q \wedge \neg p)$ should be true. From axiom A1,

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] \wedge \neg[(p \wedge q) \wedge (q \wedge \neg p)]$

$\Rightarrow[(p \wedge q) \vee (q \wedge \neg p)] \wedge \neg[F] \Rightarrow [(p \wedge q) \vee (q \wedge \neg p)]$

If (p starts q) = T and p = T, then

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] \wedge \neg[(p \wedge q) \wedge (q \wedge \neg p)] = T$

$\Rightarrow[(T \wedge q) \vee (q \wedge F)] = T$

$\Rightarrow q = T$

If (p starts q) = T and p = F, then

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] = T$

$\Rightarrow [(F \wedge q) \vee (q \wedge T)] = T$

$\Rightarrow q = T$

If (p starts q) = T and q = T, then

(p starts q) $\Rightarrow$ $[(p \wedge q) \vee (q \wedge \neg p)] = T$
$\Rightarrow$ $[(p \wedge T) \vee (T \wedge \neg p)] = T$
$\Rightarrow$ $[(p \wedge T) \vee (T \wedge \neg p)] = T$
$\Rightarrow$ $[p \vee \neg p] = T$
$\Rightarrow$ $p = T$ or $p = F$

If (p starts q) = T and q = F, then

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] = T$
$\Rightarrow$ $[(p \wedge F) \vee (F \wedge \neg p)] = T$
$\Rightarrow F \vee F = T$

This cannot happen. That is our assumption q = F is wrong. Hence if (p starts q) is true, always q should be true.

If (p starts q) = T and p = q = T, then

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] = T$
$\Rightarrow$ $[(T \wedge T) \vee (T \wedge F)] = T$
$\Rightarrow T = T$

That is, both p and q can be true at the same time when (p starts q) is true.

If (p starts q) = T and p = q = F, then

(p starts q) $\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] = T$
$\Rightarrow$ $[(F \wedge F) \vee (F \wedge T)] = T$
$\Rightarrow F = T$

That is, both p and q cannot be true at the same time when (p starts q) is true.

These show that there can only be two possible occurrences of events when (p starts q) is true: p and q both occur or only q occurs. These are the required truth states for (p starts q) or (p ends q) to occur. TAND connects these events in chronological order. Depending on the order of the occurrences either (p starts q) or (p ends q) will occur.

p during q $= (q \wedge \neg p) \prod (p \wedge q) \prod (q \wedge \neg p)$

(p during q) is true iff $(q \wedge \neg p)$ is true and then $(p \wedge q)$ is true and then $(q \wedge \neg p)$ is true. The temporal relation (p during q) can be represented as $(q \wedge \neg p) \prod (p \wedge q) \prod (p \wedge q) \prod (q \wedge \neg p)$. Because $(p \wedge q) \prod (p \wedge q)$ equals $(p \wedge q)$. Let $(p \wedge q)$ occur during time zone TL. The TL can be divided into two time zones: TL1 and TL2 such that TL1 + TL2 equals TL and TL1 follows TL2. Then $(p \wedge q)$ occur in time zone TL1 *and then* occur in time zone TL2. Therefore $(p \wedge q)$ can be expressed as $(p \wedge q) \prod (p \wedge q)$ without making any errors.

Then
(p during q) $= (q \wedge \neg p) \prod (p \wedge q) \prod (p \wedge q) \prod (q \wedge \neg p)$
$= [(q \wedge \neg p) \prod (p \wedge q)] \prod [(p \wedge q) \prod (q \wedge \neg p)] = X \prod Y$
if $[(q \wedge \neg p) \prod (p \wedge q)] = X$ and $[(p \wedge q) \prod (q \wedge \neg p)] = Y$.

Then $X = (q \wedge \neg p) \prod (p \wedge q)$
$\Rightarrow [(q \wedge \neg p) \vee (p \wedge q)] \wedge \neg[(q \wedge \neg p) \wedge (p \wedge q)]$
$\Rightarrow (q \wedge \neg p) \vee (p \wedge q)$ and
$Y = (p \wedge q) \prod (q \wedge \neg p)$
$\Rightarrow [(p \wedge q) \vee (q \wedge \neg p)] \wedge \neg[(p \wedge q) \wedge (q \wedge \neg p)]$
$\Rightarrow (p \wedge q) \vee (q \wedge \neg p) = (q \wedge \neg p) \vee (p \wedge q)$.

This means, it is sufficient to consider either only X or Y. X represents (p ends q) and Y represents(p starts q). So(p during q) can be represented as a combination of (p ends q) and (p starts q). That is (p during q) = (p ends

q) TAND (p starts q). It is shown in the earlier section that only q occurs *and then* both p and q occur are the required events for (p ends q) to occur. The reverse sequence of events, that is both p and q occur first *and then* only q occur are the necessary order of events for the event (p starts q) to occur. Therefore it can be concluded that the required order of events (p during q) to occur is, first only q occurs *and then* both p and q occur *and then* only q occurs.

p equals q $= (p \wedge q)$

(p equals q) is true, iff p and q are both true at the same time. Since standard logical connective AND is redefined to represent only concurrent situations the required representation is $(p \wedge q)$. Throughout this section it is proved that TAND can be used to represent temporal relations suggested by J. F. Allen[Allen 1983] uniquely and properly. Figure 3.4 shows them graphically.

3.3.4 Representation of temporal operators using TAND

The logical temporal connective TAND is capable of representing other temporal operators available in temporal logic. Even though there are several temporal operators such as **sometimes in the past**, **always in the past**, **sometimes in the future**, **always in the future**, **next**, **last**, **until** and **since** available in the literature, Barringer et al. [Barringer 1996] show that all temporal operators can be represented using **Until** and **Since** operators. Therefore this section presents how TAND can be used to represent **Until** and **Since** operators.

Let us first consider what $M, i \models p$ is. It means that formula p will be interpreted in model M at the state with index i [Torsun 1995][Barringer 1996]. In other words, it says that formula p holds in model M at state i or p is true at state i in model M. States are defined over the discrete linear time model. Detailed explanation about this can be found in the next chapter. This section simply employs the definition of $M, i \models p$.

p *Until* q = True $\prod$ ($\neg$q $\Rightarrow$ (p $\prod$ q))

Proof

Case 1.

M, I $\vdash$ p *Until* q iff there is some k > i such that M, k $\vdash$ q and for every j, if i < j < k then M, j $\vdash$ p (By the definition of *Until* **)**. That is,

M, i $\vdash$ p *Until* q $\Rightarrow$ there is some k>i such that M, k $\vdash$ q and for every j, if i < j < k then M, j $\vdash$ p.

$\Rightarrow$[M, i+1 $\vdash$ q or

M, i+1 $\vdash$ p and M, i+2 $\vdash$ q or

M, i+1 $\vdash$ p and M, i+2 $\vdash$ p and M, i+3 $\vdash$ q or

….

….

M, i+1 $\vdash$ p and M, i+2 $\vdash$ p and ………. and M, k-1 $\vdash$ p and M, k $\vdash$ q].

$\Rightarrow$ [M, i+1 $\vdash$ q or

M, i+1 $\vdash$ (p $\prod$ q) or

M, i+1 $\vdash$ (p $\prod$ p $\prod$ q) or

….

….

M, i+1 $\vdash$ (p $\prod$ p $\prod$…………………….. $\prod$ p $\prod$ q)].

$\Rightarrow$ [M, i+1 $\vdash$ q or

M, i+1 $\vdash$ (p $\prod$ q) or

M, i+1 $\vdash$ (p $\prod$ q) or

….

….

M, i+1 $\vdash$ (p $\prod$ q)]; Since p $\prod$ p = p.

$\Rightarrow$ M, i+1 $\vdash$ q or

M, i+1 $\vdash$ [(p $\prod$ q) $\vee$ (p $\prod$ q) $\vee$ ……………$\vee$ (p $\prod$ q)].

$\Rightarrow$ M, i+1 $\vdash$ q or M, i+1 $\vdash$ (p $\prod$ q).

$\Rightarrow$ M, i+1 $\vdash$ [q $\vee$ (p $\prod$ q)].

$\Rightarrow$ M, i+1 $\vdash$ [$\neg$q $\Rightarrow$ (p $\prod$ q)].

$\Rightarrow$ M, i $\models$ True and M, i+1 $\models$ [$\neg$q $\Rightarrow$ (p Π q)].
$\Rightarrow$ M, i $\models$ True Π [$\neg$q $\Rightarrow$ (p Π q)].
$\Rightarrow$ True Π [$\neg$q $\Rightarrow$ (p Π q)].

Case 2.
M, i $\models$ True Π [$\neg$q $\Rightarrow$ (p Π q)]
$\Rightarrow$ M, i $\models$ True and M, i+1 $\models$[$\neg$q $\Rightarrow$ (p Π q)].
$\Rightarrow$ M, i+1 $\models$ [$\neg$q $\Rightarrow$ (p Π q)].
$\Rightarrow$ M, i+1 $\models$ [q $\vee$ (p Π q)].
$\Rightarrow$ M, i+1 $\models$ q or M, i+1 $\models$ (p Π q).
$\Rightarrow$ M, i+1 $\models$ q or
M, i+1 $\models$ [(p Π q) $\vee$ (p Π q) $\vee$ ……………$\vee$ (p Π q)].
$\Rightarrow$ [M, i+1 $\models$ q or
M, i+1 $\models$ (p Π q) or
M, i+1 $\models$ (p Π q) or
….
….
M, i+1 $\models$ (p Π q)].
$\Rightarrow$ [M, i+1 $\models$ q or
M, i+1 $\models$ (p Π q) or
M, i+1 $\models$ (p Π p Π q) or
….
….
M, i+1 $\models$ (p Π p Π………………….. Π p Π q)]; Since p = p Π p.
$\Rightarrow$(M, i+1 $\models$ q or
M, i+1 $\models$ p and M, i+2 $\models$ q or
M, i+1 $\models$ p and M, i+2 $\models$ p and M, i+3 $\models$ q or
….
….
M, i+1 $\models$ p and M, i+2 $\models$ p and ……… and M, k-1 $\models$ p and M, k $\models$ q).
$\Rightarrow$ there is some k > i such that M, k $\models$ q and for every j, if i<j<k then M, j $\models$ p.
$\Rightarrow$ p *Until* q.

p *Since* q = $(\neg q \Rightarrow q \prod p) \prod \text{True}$

Proof

Case 1.

M, i ├ p *Since* q iff there is some k < i such that M, k ├ q and for every j, if k < j < i then M, j ├ p. That is,

M, i ├ p *Since* q ⇒ there is some k < i such that M, k ├ q and for every j, if k < j < i then M, j ├ p.

⇒(M, i-1 ├ q or

M, i-1 ├ p and M, i-2 ├ q or

M, i-1 ├ p and M, i-2 ├ p and M, i-3 ├ q or

….

….

M, i-1 ├ p and M, i-2 ├ p and ………. and M, k+1 ├ p and M, k ├ q).

⇒[M, i-1 ├ q or

M, i-1 ├ (q ∏ p) or

M, i-1 ├ (q ∏ p ∏ p) or

….

….

M, i-1 ├ (q ∏ p ∏…………………….. ∏ p ∏ p)].

⇒[M, i-1 ├ q or

M, i-1 ├ (q ∏ p) or

M, i-1 ├ (q ∏ p) or

….

….

M, i-1 ├ (q ∏ p)]; Since p ∏ p = p.

⇒ M, i-1 ├ q or

M, i-1 ├ [(q ∏ p) ∨ (q ∏ p) ∨ ……………∨ (q ∏ p)].

⇒ M, i-1 ├ q or M, i-1 ├ (q ∏ p).

⇒ M, i-1 ├ [q ∨ (q ∏ p)].

⇒ M, i-1 ├ [¬q ⇒ (q ∏ p)].

$\Rightarrow$ M, i $\vdash$ True and M, i-1 $\vdash [\neg q \Rightarrow (q \prod p)]$.

$\Rightarrow$ M, i $\vdash$ $[\neg q \Rightarrow (q \prod p)] \prod$ True.

$\Rightarrow$ $[\neg q \Rightarrow (q \prod p)] \prod$ True.

Case 2.

M, i $\vdash$ $[\neg q \Rightarrow (q \prod p)] \prod$ True

$\Rightarrow$ M, i $\vdash$ True and M, i-1 $\vdash [\neg q \Rightarrow (q \prod p)]$.

$\Rightarrow$ M, i-1 $\vdash [\neg q \Rightarrow (q \prod p)]$.

$\Rightarrow$ M, i-1 $\vdash [q \vee (q \prod p)]$.

$\Rightarrow$ M, i-1 $\vdash$ q or M, i-1 $\vdash$ $(q \prod p)$.

$\Rightarrow$ M, i-1 $\vdash$ q or

M, i-1 $\vdash [(q \prod p) \vee (q \prod p) \vee \ldots\ldots\ldots\ldots\ldots \vee (q \prod p)]$.

$\Rightarrow$[M, i-1 $\vdash$ q or

M, i-1 $\vdash (q \prod p)$ or

M, i-1 $\vdash (q \prod p)$ or

….

….

M, i-1 $\vdash (q \prod p)$].

$\Rightarrow$[M, i-1 $\vdash$ q or

M, i-1 $\vdash (q \prod p)$ or

M, i-1 $\vdash (q \prod p \prod p)$ or

….

….

M, i-1 $\vdash (q \prod p \prod \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots p \prod p)$]; Since $p = p \prod p$.

$\Rightarrow$(M, i-1 $\vdash$ q or

M, i-1 $\vdash$ p and M, i-2 $\vdash$ q or

M, i-1 $\vdash$ p and M, i-2 $\vdash$ p and M, i-3 $\vdash$ q or

….

….

M, i-1 $\vdash$ p and M, i-2 $\vdash$ p and ………… and M, k+1 $\vdash$ p and M, k $\vdash$ q).

$\Rightarrow$ there is some $k < i$ such that M, k $\vdash$ q and for every j, if $k < j < i$ then M, j $\vdash$ p.

$\Rightarrow$ p *Since* q.

Since *Until* and *Since* operators can be defined using TAND and all other temporal operators can be represented using *Until* and *Since* operators as shown by Barringer et al.[Barringer 1996], all the point based temporal operators can be defined using TAND.

Sometimes in the future A = True $\prod$ [$\neg$A $\Rightarrow$ (True $\prod$ A)]

Proof
Sometimes in the future A
= True *Until* A ; by the definition [Barringer 1996].
= True $\prod$ [$\neg$A $\Rightarrow$ (True $\prod$ A)]; replacing p by True and q by A
in (p *Until* q).

Always in the future A = $\neg$[True $\prod$ (A $\Rightarrow$ (True $\prod$ $\neg$A))]

Proof
Always in the future A
= $\neg$(*Sometimes in the future* $\neg$A) ; by the definition[Barringer 1996].
= $\neg$(True *Until* $\neg$A) ; by the definition[Barringer 1996].
= $\neg$[True $\prod$ (A $\Rightarrow$ (True $\prod$ $\neg$A))] ; replacing p by True and q by $\neg$A in
(p *Until* q).

Next A = True $\prod$ ($\neg$A $\Rightarrow$ (false $\prod$ A))

Proof
Next A = False *Until* A ; by the definition[Barringer 1996].
= True $\prod$ ($\neg$A $\Rightarrow$ (false $\prod$ A)) ; replacing p by False and
q by A in (p *Until* q).

Sometimes in the past A = ($\neg$A $\Rightarrow$ A $\prod$ True) $\prod$ True

Proof

Sometimes in the past A

= True *Since* A ; by the definition[Barringer 1996].

= $(\neg A \Rightarrow A \prod \text{True}) \prod \text{True}$; replacing p by True and q by A in (p *Since* q).

Always in the past A = $\neg[(A \Rightarrow \neg A \prod \text{True}) \prod \text{True}]$

Proof

Always in the past A

= $\neg$(True *Since* $\neg$A) ; by the definition[Barringer 1996].

= $\neg[(A \Rightarrow \neg A \prod \text{True}) \prod \text{True}]$; replacing p by True and q by $\neg$A in (p *Since* q).

Last A = $(\neg A \Rightarrow A \prod \text{False}) \prod \text{True}$

Proof

Last A = False *Since* A ; by the definition[Barringer 1996].

= $(\neg A \Rightarrow A \prod \text{False}) \prod \text{True}$; replacing p by False and q by A in (p *Since* q).

As shown above TAND logical temporal connective is capable of representing all existing temporal operators as well as all relative temporal relations suggested by J. F . Allen [Allen 1983]. Since TAND is not based on any time models, it can rectify the weaknesses of both point based and interval based temporal models. It is simple and easy to use. Further it can be used with any time model; purely point based, purely interval based or point-interval based time models. Application of TAND is not limited to temporal logic but it can be used where the standard logical connective AND is used. Since all the temporal operators and relations can be defined using TAND, now we can think about a new temporal logic which employs TAND instead of standard temporal operators or relations to represent temporal information.

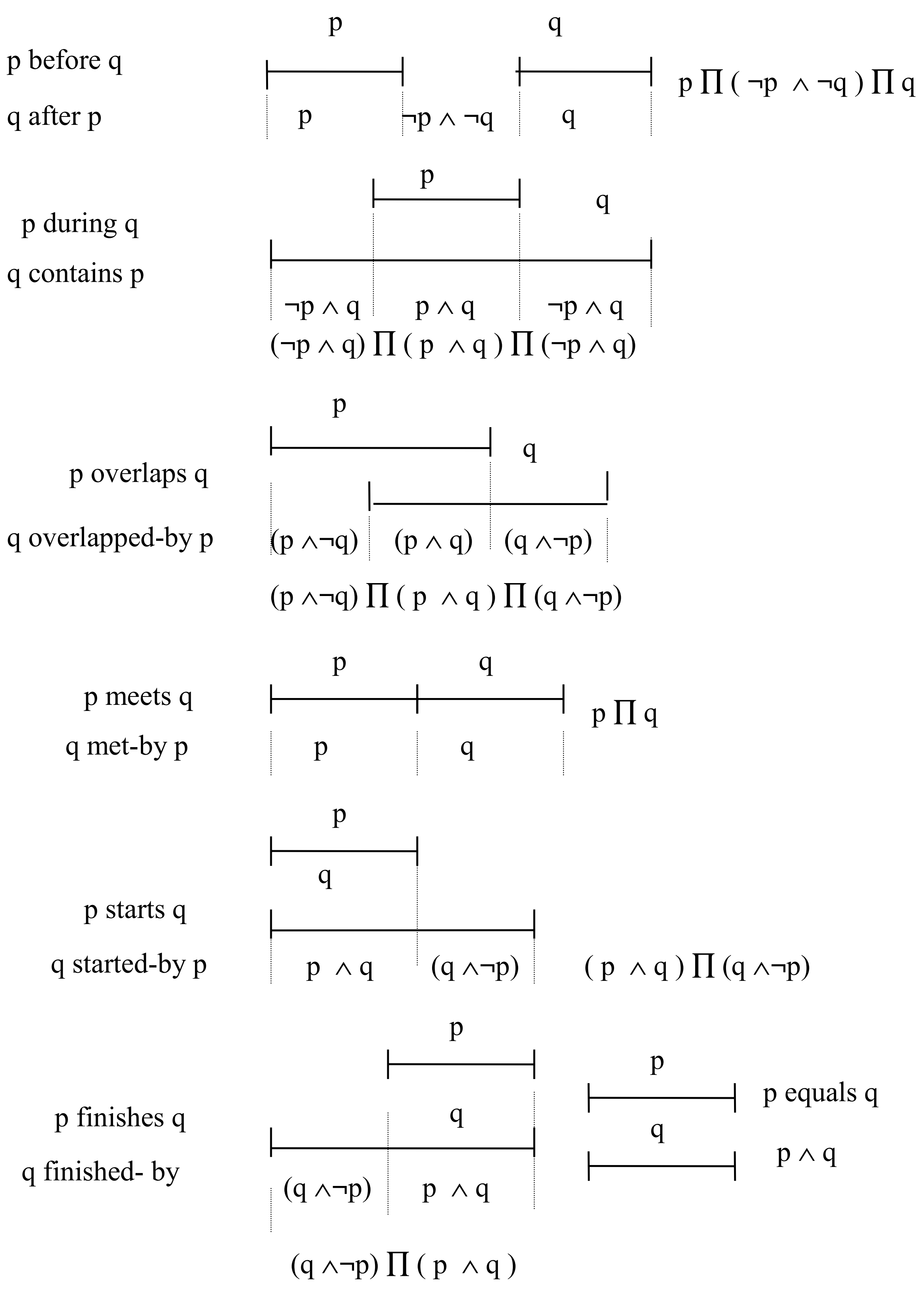


Figure 3.4 TAND versions of Allen's temporal relations

CHAPTER IV

TEMPORAL LOGIC

This chapter discusses about the first-order temporal logic(FTL), the temporal logic USF and the temporal logic of actions (TLA) which are used in knowledge based specifications, and their problems. The last section introduces the new temporal logic TANDTL. The temporal logic TANDTL employs TAND connective to represent temporal knowledge. Therefore TANDTL does not have any temporal operators or temporal relations instead it has TAND connective in addition to standard logical connectives.

4.1 Conventional temporal logic

Representing and reasoning about time play a critical role in many areas of studies. In computer science, temporal logic[Kroger 1987][Manna 1995][Torsun 1995][Barringer 1996] has been used in concurrent systems, real-time systems, specifications, program verification, temporal database, planning expert systems and natural languages. Temporal logic has shown to be a powerful and flexible formalism, capable of representing and reasoning in these areas. In this section, conventional temporal logic, its weakness in software requirements specifications and proposed temporal logic will be discussed.

4.1.1 First-order temporal logic

First order temporal logic can be considered as a extension of first order logic with temporal operators.

Syntax

A language of first order temporal logic consists of the following set of symbols:

1. A set P of predicate symbols.
2. A set F of function symbols.
3. The equality symbol =.
4. A set V of variable symbols.
5. A set C of constant symbols.
6. A set Q of propositions.
7. The set of logical connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$).
8. The set of temporal operators (O, □, ◊,U, W, ●, ■, ◆, S, Z).
9. The set of quantifiers ($\forall$, $\exists$).

These sets are disjoint. Given a first order temporal language, formulae of the language are constructed using the following rules:

1. All variable symbols are terms.
2. If f is a function symbol of arity, k, k ≥0 and t1, …, tk are terms, then f(t1,…..,tk) is a term, f() is a constant.
3. If p is a predicate symbol of arity, k, k ≥0 and t1, …, tk are terms, then p(t1, ……, tk) is an atomic formula. If k=0 then p() is a proportional constant.
4. If t1 and t2 are terms, then t1=t2 is an atomic formula.

The formation rules for FTL are

1. All atomic formula are formulae.
2. Formulae are constructed from other formulae by application of connectives and temporal operators.
3. If x is a variable symbol and A is a formula, then ∀xA, ∃xA are formulae.

Semantics

A formal semantics is described in terms of Kripke's possible worlds [Torsun 1995][Fisher 1995]. The possible worlds are set of states S in times which are temporally related via the accessibility relation R. More formally a frame is defined by as a pair <S, R> where S is a non-empty set of possible states and R is a binary relation over $R \subseteq S \times S$, called the accessibility relation. Given a first order temporal language L, a model M is a tuple M = (S, R, D, ∏,ν) where

- (S,R) is a linear temporal frame where S is a non-empty set of possible states; R is a binary relation over $R \subseteq S \times S$, called the accessibility relation.
- D is a non-empty set, called the interpretation domain. Domain D is constant.
- ∏ is an interpretation function for variables, which maps each variable $x \in V$ (a set of variables) to an element $\prod(x) \in D$.
- ν is a valuation function for constants, which maps:

  each function symbol $f \in F$(a set of functions) of arity n to a relation ν(f) from D^n into D;

each predicate symbol $p \in P$ (a set of predicate symbols) of arity m to a relation $\nu(p)$ from D^m into the set $\{T, F\}$;

each constant $c \in C$ (a set of constants) to an element $\nu(c) \in D$;

each proposition $q \in Q$ (a set of propositions) to a truth value $\nu(q) \in \{T,F\}$.

Given a temporal interpretation $M = (S, R, D, \Pi,\nu)$ and state $s \in S$ (index i indicates state i, future times are larger than i, past times are less than i), the following rules associate a truth value with each formula A, and associate an element of D with each term t.

An interpretation for the logic is defined as a pair <M,i>, where M is the model and i is the index of the state at which the temporal statement is to be interpreted. $<M, i> \vdash A$ means that "A is true in the model M at the state i".

•For terms:

$<M, x> = \Pi(x)$ for $x \in V$; says that the interpretation of each variable x in the model M at a given state s is $\Pi(x)$.

$<M, c> = \nu(c)$ for $c \in C$.

$<M, f(t_1,\ldots,t_n)> = \nu(f)(<M,t_1>,\ldots,<M,t_n>)$ for $f \in F$ and $t_1,\ldots,t_n$ are terms.

Interpretation for well-formed formulae is defined in terms of the satisfaction relation F between models and formulae.

- For atomic formulae:

$<M, i> \vdash p(t_1, \ldots,t_m)$ iff $\nu<i,p>(<M,t_1>,\ldots,<M,t_m>)=T$,

$<M, i> \vdash t_1=t_2$ iff $<M,t_1> = <M,t_2>$.

If $d_1,\ldots d_n$ are elements of D, then $Mo<x_1\leftarrow d_1,\ldots x_n\leftarrow d_n>$ denotes the model obtained from M by modifying its assignment function to map the variable symbols $x_1,\ldots,x_n$ to $d_1,\ldots d_n$ respectively.

Given a first order temporal language FTL, a model structure $M = (S, R, D, \alpha, I)$, evaluation functions ρ and Π, the interpretation rules for FTL are as follows.

- For terms:

$\rho(M, x) = \alpha(x)$,

$\rho(M, f(t_1,\ldots,t_n)) = I(s,f)\ (\rho(M, t_1),\ldots..\rho(M,t_n))$.

- For atomic formulae:

  $<M, \rho, \Pi, s_i> \models p(t_1,\ldots.t_n)$ iff $I(s,f) <(\rho(M, t_1),\ldots.\rho(M,t_n)) \in \Pi(s_i)$,

  $<M, \rho, \Pi, s_i> \models t_1=t_2$ iff $\rho(M, t_1) = \rho(M,t_2)$.

- For connective: If A and B are formulae then

  $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ and $(A \Leftrightarrow B)$are interpreted as follows:

  $<M, i> \models T$

  $<M, i> \models \neg A$ iff not $<M, i> \models A$

  $<M, i> \models A \wedge B$ iff $<M, i> \models A$ and $<M, i> \models B$

  $<M, i> \models A \vee B$ iff $<M, i> \models A$ or $<M, i> \models B$.

  The semantics for $\Rightarrow$ and $\Leftrightarrow$ defined in terms of $\neg$ and $\vee$.

- For temporal operators: If A and B are formulae then

  OA (Next A), □A (Always in the future A), ◊A (Sometimes in the future A), AUB (A until B), ●A (Last A), ■A (Always in the past A), ◆A (Sometimes in the past A), ASB (A since b) are interpreted as follows:

  $<M, i> \models OA$ iff $<M, i+1> \models A$

  $<M, i> \models \Box A$ iff $\forall j \geq i <M, j> \models A$

  $<M, i> \models \Diamond A$ iff $\exists j \geq i <M, j> \models A$

  $<M, i> \models AUB$ iff $\exists j \geq i <M, j> \models B$ and

  $\forall k$ if $(i \leq k < j)$ then $<M, k> \models A$

  $<M, i> \models$ ●A iff $i = 0$ or $<M, i-1> \models A$

  $<M, i> \models$ ■A iff $\forall j\ 0 \leq j < i$, $<M, j> \models A$

  $<M, i> \models$ ◆A iff $\exists j\ 0 \leq j < i$, $<M, j> \models A$

  $<M, i> \models ASB$ iff $\exists j\ 0 \leq j < i$, $<M, j> \models A$ and

  $\forall k$ if $(i < k < j)$ then $<M, k> \models A$

- For quantifiers:
  $<M, i> \models \forall x.A$ iff all $d \in D[Mo<x \leftarrow d>, i \models A]$,
  $<M, i> \models \exists x.A$ iff some $d \in D[Mo<x \leftarrow d>, i \models A]$.

4.1.2 The logic USF

This logic [Barringer 1996] uses temporal operators since (S^-) and until (U^+) together with a fixed point operator Ψ. All the other temporal operators are defined using since (S^-) and until (U^+) operators. There are four types of well formed formulae: pure past formulae (talking only about the strict past), pure present formulae (talking only about the present), pure future formulae (talking only about strict future) and mixed formulae (talking about the entire flow of time).

<u>Syntax</u>

Let Q be a sufficiently large set of atoms. Let $\neg$, $\wedge$, $\vee$, $\Rightarrow$, true, false be the usual classical connectives and let S^- and U^+ be the temporal connectives and Ψ be the fixed point operator.

1. An atomic $q \in Q$ is in wff^0 (pure present wff) and in wff (well formed formula). Its set of atoms in $\{q\}$.
2. Assume A and B are wffs with atoms $\{q_1,\ldots\ldots,q_m\}$ and $\{r_1,\ldots..r_m\}$ respectively. Then $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $B\ U^+ A$ and $B\ S^- A$ are all wffs with atoms $\{q_1,\ldots q_n,r_1,\ldots..r_m\}$.
   a. If both A and B are in wff^0 or wff^+(pure future wff) then $B\ U^+ A$ is in wff^+.
   b. If both A and B are in wff^0 or wff^- (pure past wff), then $B\ S^- A$ is in wff^-.
   c. If both A and B are in wff*, then so are $A \wedge B$, $A \vee B$, $A \Rightarrow B$, where wff* is one of wff^+, wff^-, wff^0.
3. $\neg A$ is also a wff and it is of the same type as A with the same atoms as A.
4. True (Truth) and false (Falsity) are wffs in wff^0 with no atoms.

5. If A is a wff in wff^{-} with atoms $\{q,q_1,\ldots,q_n\}$ then $(\Psi q)A$ is a wff^{-} with the atoms $\{q_1,\ldots,q_n\}$.

Semantics

Let D be a non-empty set, called the domain, and *g* be a function assigning the following:

1. For each m-place function symbol *f* and each $n \in N$ a function $g(n,f) : D^m \Rightarrow D$.
2. For each variable x and each $n \in N$, an element $g(n, x) \in D$.
3. For each m-place function symbol Q and each $n \in N$ a function $g(n, Q): D^m \Rightarrow \{0, 1\}$.

The function g can be extended to a function $g(n, A)$, giving a value in $\{0, 1\}$ for each wff $A(x_1,\ldots x_n)$ of the predicate USF [Barringer 1996] as follows:

1. $g(n, f(t_1,\ldots,t_m)) = g(n, f)(g(n, t_1),\ldots,g(n, t_m))$.
2. $g(n, Q(t_1,\ldots,t_m)) = g(n, Q)(g(n, t_1),\ldots,g(n, t_m))$.
3. $g(n, A \wedge B) = 1$ iff $g(n, A) = 1$ and $g(n, B) = 1$.
4. $g(n, A \vee B) = 1$ iff either $g(n, A) = 1$ or $g(n, B) = 1$ or both.
5. $g(n, A \Rightarrow B) = 1$ iff either $g(n, A) = 0$ or $g(n, B) = 1$ or both.
6. $g(n, \neg A) = 1$ iff $g(n, A) = 0$.
7. $g(n, \text{true}) = 1$ and $g(n, \text{false}) = 0$ for all n.
8. $g(n, B\ U^{+} A) = 1$ iff for some $m > n$, $g(m, A) = 1$ and for all $n < k < m$, $g(k, B) = 1$.
9. $g(n, B\ S^{-} A) = 1$ iff for some $m < n$, $g(m, A) = 1$ and for all $m < k < n$, $g(k, B) = 1$.
10. $g(n, \forall x\ A(x)) = 1$ for a variable x iff for all g' such that g' gives the same values as *g* to all function symbols and all predicate symbols and all variables different from x, we have $g'(n, A(x)) = 1$.
11. $g(n, \exists x\ A(x)) = 1$ for a variable x iff for some g' such that g' gives the same values as *g* to all function symbols and all predicate symbols and all variables different from x, we have $g'(n, A(x)) = 1$.
12. Let $(\Psi q)\ A(q, q_1,\ldots,q_m)$ be a pure past formula of USF, and let $B_i \in wffV^{*}_{i}$ for $i = 1,\ldots,m$. To define $g(n, A')$ where $A' = (\Psi q)\ A(q, B_1,\ldots,B_m)$, first choose an assignment h such that $h(qi) = \{n \in N \mid$

$g(n,B_i) = 1\}$. Then define $g(n, A') = 1$ iff $n \in h((\Psi q)\ A(q, q_1,\ldots\ldots,q_m))$.

4.1.3 The Temporal Logic of Actions

The temporal logic of actions (TLA) is a logic for specifying and reasoning about concurrent systems introduced by Leslie Lamport [Lamport 1994]. It combines two logics : a logic of actions and conventional temporal logic. The logic of actions consists of values, variables, states, state functions, predicates and actions while conventional temporal logic consists standard logical connectives and point based temporal operators. The temporal logic of actions is obtained by letting the elementary temporal formula in conventional temporal logic be actions in logic of actions.

Values	: data items (“true” and “false” are not included).
Variables	: unprimed variables (refer to old state) and primed variables (refer to new state).
States	: assignment of values to variables.
State functions	: non Boolean expressions built from variables and constant symbols (Example : $x^2 + y - 3$).
Predicates	: Boolean expression built from variables and constant symbols (Example : $x^2 = y - 3$).
Actions	: Boolean- valued expressions formed from variables, prime variables and constant symbols. (Example : $x' + 1 = y$ where y is a variable and x’ is a primed variable).

Example : Representing a simple program using TLA [Lamport 1994].
A simple program, written in a conventional language:

```
var natural x, y = 0;
do < true → x := x + 1>
```

$<\text{true} \rightarrow y := y + 1>$ od

This program is written in a conventional language, using Dijkstra's do construct, with angle brackets enclosing operations that are assumed to be atomic. An execution of this program begins with x and y both zero, and repeatedly increments either x or y (in a single operation), choosing non deterministically between them [Lamport 1994].

TLA formula ϕ describing above program:

$$\begin{aligned}
\text{Init}_{\phi} &\triangleq (x = 0) \wedge (y = 0) \\
M1 &\triangleq (x' = x + 1) \wedge (y' = y) \\
M2 &\triangleq (y' = y + 1) \wedge (x' = x) \\
M &\triangleq M1 \vee M2
\end{aligned}$$

$$\phi \triangleq \text{Init}_{\phi} \wedge \quad M$$

(The symbol $\triangleq$ means equal by definition)

The execution of the program is represented by ϕ = true. The predicate Init_{ϕ} asserts the initial conditions, that x and y are both zero. The semantic meaning of action M1 is a relation between states asserting that the value of x in the new state is one greater than its value in the old state, and the value of y is the same in the old and new states. Thus a M1 step represents an execution of the program's atomic operation of incrementing x. Similarly, a M2 step represents an execution of the program's other atomic operation, which increments y. The action M is defined to be the disjunction of M1 and M2, so a M step represents an execution of one program operation. ϕ asserts that the initial condition is true initially, and that every step of the behavior represents the execution of an atomic operation of the program [Lamport 1994].

4.1.4 Problems with conventional temporal logic

Temporal operators namely OA (Next A), □A (Always in the future A), ◊A (Sometimes in the future A), AUB (A until B), ●A (Last A), ■A

(Always in the past A), ◆A (Sometimes in the past A), ASB (A since b) are available in conventional temporal logic. These temporal operators employ point based time model to represent and reason about temporal knowledge. Therefore conventional temporal logic inherits the problems discussed in section 3.2.6.

4.2 TANDTL temporal logic

This research proposes a new temporal logic TANDTL by removing existing temporal operators from the first-order temporal logic [Kroger 1987][Manna 1995][Torsun 1995][Barringer 1996] and including TAND logical connective to represent temporal information . Further, the standard AND connective is redefined to represent concurrent events[Wijayarathna 1997]. This section describes the temporal logic TANDTL which can solve the above discussed problems of conventional temporal logic. The temporal logic TANDTL will be used as the formal basis for software requirements specifications language GSL and its execution.

Syntax

1. Symbols
   - The truth symbols T and F.
   - The equality symbol =.
   - The possessions symbol $\in$.
   - A set C of constant symbols a, b, c, ….
   - A set V of variable symbols u, v, w, x, y, z, …..
   - A set F of function symbols f, g, h, ….. Each function symbol has an a positive integer called, its arity, indicating number of arguments.
   - A set P of predicate symbols p, q, r, ….. Each predicate symbols also has an arity.
   - A set Q of propositions.
   - The set of standard logical connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$).
   - The temporal logical connective $\prod$.

- The set of quantifiers (∀, ∃).

The constants and variables will represent objects, and the functions and predicates will represent functions and relations in those objects.

2. Terms

   The terms are the objects of the logic. They are built up according to following rules;
   - The constants a, b, c are terms.
   - The variables x, y, z are terms.
   - If t1, t2, …….tn are terms (n ≥ 1) then f(t1, t2, …..,tn) is also a term. Function f is a n-arity function.
3. Atomic formulae or atoms

   Atoms represents the relation between objects and are constructed according to following rules;

   - The truth symbols T (true) and F (false) are atoms.
   - If t1, t2, …….tn are terms (n ≥ 1) then P(t1, t2, …..,tn) is an atom. Predicate P is a n-arity predicate.

4. Formulae or sentences

   The sentences of the logic are built from atoms according to the following rules;
   - Every atom is a sentence.
   - If A and B are sentences then so are ¬A, A ∧ B, A ∨ B, A ⇒ B, A ⇔B and A ∏ B.
   - If x is a variable, then ∀x (A) and ∃x (A) are sentences.

5. Priority order of the connectives

   The connectives are listed from highest to lowest.
   - ¬, ∀, ∃
   - ∨

- $\wedge$
- $\prod$
- $\Rightarrow, \Leftrightarrow$

6. Expressions

   An expression is either a term or a formula. Thus x, f(x) and p(x) are expressions.

7. Sub-term, Sub-formula and sub-expression
   - Sub-term is a intermediate term which is used to build a term or a formula.
   - Sub-formula is also a intermediate formula which is build a term or a formula.
   - Sub-terms and sub-formulas are called sub-expressions.

8. Free and bound variables

   The occurrence of a variable x in an expression is bounded if x is within the scope of quantifiers $\forall$x and $\exists$x. Otherwise it is free.

<u>Semantics</u>

Let us assume that A and B are formulae. Then

- T is always true.
- F is always false.
- $\neg$A is true if and only if A is false.
- A $\wedge$ B is true if and only if both A and B are true at the same time. That is A and B are true simultaneously.
- A $\vee$ B is true if and only if A is true or B is true, and false otherwise (if both A and B are false at the same time).
- A $\prod$ B is true if and only if A $\wedge$ B is false (that is A and B cannot be true simultaneously) and A is true and then B is true.
- A $\Rightarrow$ B is true if and only if A is false or B is true.

- $A \Leftrightarrow B$ is true if and only if both A and B have same logical truth value.
- The expression $X \in A$ express that X belongs to A. If $X \in A$ and $A \in B$ then it implies that $X \in B$. That is $(X \in A) \wedge (A \in B) \Rightarrow (X \in B)$.

Note that TANDTL temporal logic is not based on any fixed time model. Instead all temporal aspects are represented using relative time. That is to say whether two events A and B occur concurrently or not. If not, one event should follow the other. However, if we need, point based version of TANDTL temporal logic can be defined using Kripke's possible worlds[Kripke 1959].

<u>Semantics of point based version of TANDTL</u>

- T is true at any given time t.
- F is false at any given time t.
- $\neg A$ is true at any given time t if and only if A is false at time t.
- $A \wedge B$ is true at any given time t if and only if both A and B are true at time t. That is A and B are true simultaneously.
- $A \vee B$ is true at any given time t if and only if A is true at time t or B is true at time t, and false otherwise that (if both A and B are false at time t).
- $A \prod B$ is true if and only if A is true at time t and B is true at time t+1. The informal meaning of $A \prod B$ will be "A next B". In the same way if B is true at time t and A is true at time t-1, then $A \prod B$ is true at time t.
- $A \Rightarrow B$ is true if and only if A is false or B is true.
- $A \Leftrightarrow B$ is true if and only if both A and B have same logical truth value.

Therefore temporal logic TANDTL can be used with both point based and interval based time models. In interval based models, $A \prod B$ means "A meets B" while in point based models it means "A next B". Figure 4.1 shows these temporal relations graphically. Therefore we will be able to execute a specifications based on TANDTL in a point based computer while providing the expressive power to represent the incomplete relative temporal knowledge.

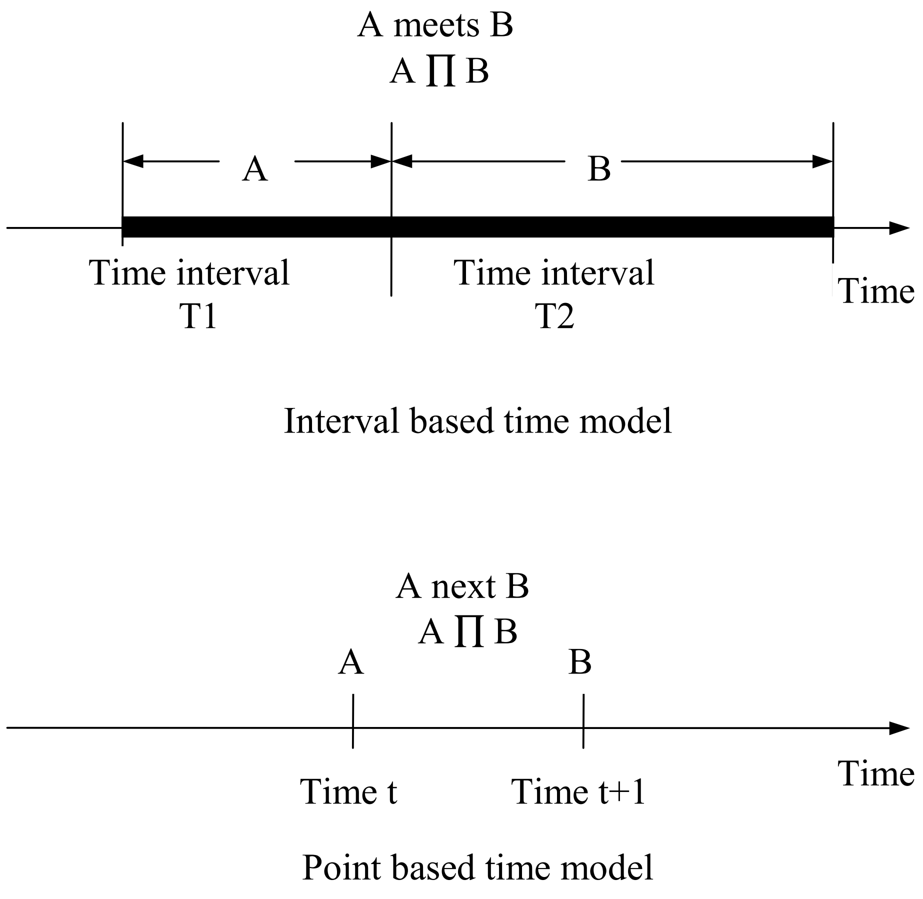


Figure 4.1 TAND with point based and interval based time models

4.3 Inconsistency proofs based on resolution

Temporal logic has been used in various applications such as program verification, specification, synthesis of concurrent systems, synthesis of robot plans, verification of hardware devices and verification of reactive systems [Abadi 1990][Manna 1995][Torsun 1995][Tuzhilin 1995]. Some of these applications require proofs within temporal logic[Abadi 1990].

4.3.1 Classical resolution

Resolution is a simple proof technique used in classical logic to prove the inconsistency of logical formulae [Torsun 1995]. The use of the resolution rule for proving inconsistency is restricted to formulae in conjunctive normal form. A proposition A is in conjunctive normal form if it is a conjunction ($C_1 \wedge \ldots \wedge C_m$) of disjunction of $C_i = B_{i,1} \vee \ldots\ldots \vee B_{i,n}$ where each B_{ij} is a literal, that is a propositional symbol or the negation of a propositional symbol. The empty clause is the only inconsistent clause; it is represented by F [Torsun 1995]. The inconsistency of the set S can be checked by generating logical consequences of S ending up with F. A very simple reasoning scheme will be used to generate logical consequences. Let A, B and C be formulae. Let us suppose that both formulae ($A \vee C$) and ($B \vee \neg C$) are true. If C is also true, then B is obviously true; otherwise, if C is false, then A is true. In both cases, ($A \vee B$) is true. This leads to the rule

$$\{ A \vee C, B \vee \neg C \} \vdash A \vee B,$$

which can also be written as

$$\{ \neg C \Rightarrow A , C \Rightarrow B \} \vdash A \vee B.$$

In the special case where C is a single proposition and where A and B are clauses, this rule is called the resolution rule [Torsun 1995]. Let S_1 and S_2 be two clauses belonging to the normal form S and let L be a literal. If $L \in S_1$ and $\neg L \in S_2$, then the inconsistency of the set S of clauses can be proved by the following algorithm:

```
While    $F \notin S$,
         select L, $S_1$, $S_2$ such that $L \in S_1$ and $\neg L \in S_2$;
         compute resolvent R;
         replace S by $S \cup \{R\}$.
```

Resolvent R is obtained by $\{ S_1 - L \} \cup \{S_2 - \neg L\}$.

Example : Let us prove the set of clauses $S = \{ P \vee Q, P \vee R, \neg Q \vee \neg R, \neg P\}$ is inconsistent.

It is convenient to number the clauses.

(1) $P \vee Q$

(2) $P \vee R$

(3) $\neg Q \vee \neg R$

(4) $\neg P$

Then applying the above algorithm,

(5) Q	(1,4) ; $S_1 = P \vee Q$, $S_2 = \neg P$, $L = P$.
(6) R	(2,4)
(7) $\neg Q$	(3,6)
(8) F	(5,7)

Since F is obtained at the end of the process, the set of clauses S is inconsistent.

4.3.2 Non-clausal resolution for propositional logic

Transforming a formula into normal form can be a lengthy process and the use of the resolution principle for proving inconsistency is restricted to formulae in conjunctive normal form. Non-clausal resolution technique allows us to extend the resolution proof machinery to arbitrary formulae. This allows one to use the resolution method as a proof technique, based on the deduction principle, even if the hypotheses and the negation of the conclusions are not clauses.

Deduction principle:

The formula C is a logical consequences of the finite set S if and only if $S \cup \{\neg C\}$ is inconsistent. A set is inconsistent if and only if F (false) is a logical consequence of it. Formally the deduction principle is expressed as
$\{H_1, \ldots\ldots, H_n\} \vdash C \Leftrightarrow \{H_1, \ldots\ldots, H_n, \neg C\} \vdash F$ (false) [Torsun 1995].

Notation : $\perp P(A_1, A_2) = {}^{A1P\vdash F} \vee A_2{}^{P\vdash T}$ where $A_1{}^{P\vdash Q}$ denotes the formula obtained by replacing each occurrences of P by Q in A.

Example : $\{A_1 : (P \wedge R) \vee (Q \wedge \neg R), A_2 : (\neg P \wedge R) \vee (\neg Q \wedge \neg R)\}$

(1) $(P \wedge R) \vee (Q \wedge \neg R)$	A_1
(2) $(\neg P \wedge R) \vee (\neg Q \wedge \neg R)$	A_2
(3) $(Q \wedge \neg R) \vee (\neg Q \wedge \neg R)$	$\perp P(1, 2)$

(4) $(P \wedge R) \vee (\neg P \wedge R)$ $\perp Q(1, 2)$

(5) F (false) $\perp R(4, 3)$

That is logical formulae A1 and A2 are inconsistent.

4.3.3 Non-clausal resolution for first-order temporal logic

This section presents the non-clausal resolution method for first-order temporal logic introduced by Abadi and Manna [Abadi 1990] [Torsun 1995].

The non-clausal resolution rule for classical propositional logic can be stated as $A<u,\ldots,u>, B<u,\ldots.u> \longmapsto A<true> \vee B<false>$
where formulae $A<u,\ldots,u>$ and $B<u,\ldots.u>$ have common sub formula u and the resolvent can be obtained by substituting true for one or more occurrences of u in $A<u,\ldots,u>$ and false for one or more occurrences of u in $B<u,\ldots.u>$ and then taking the disjunction of their results. The expression $A<true> \vee B<false>$ is called the resolvent of A and B. The deduction rules have the form

$$u_1,\ldots..u_m \quad \longmapsto \quad v$$

can be applied to the formula S_i if the formulae $u_1,\ldots u_m$ occur as conjuncts in S_i. Then, in order to obtain s_{i+1}, the derived formula v is added to the conjunction.

This rule cannot be used in propositional or first-order temporal logic since the occurrence of u may refer to different instants of time.

Example : Consider the formula u in the formulae $\neg u$ and $\Diamond u$

If the classical resolution rule is applied,

(1) $\neg u$

(2) $\Diamond u$

(3) $\neg true \vee \Diamond false$ $\perp u(2, 1)$

However, u in $\Diamond u$ can be true at a time and false at another time. Hence , we cannot deduce that the resolvent $\neg true \vee \Diamond false$ is always false. Therefore Abadi and Manna [Abadi 1990] extend the classical non-clausal resolution method introducing several modality rules so that it can be used in propositional and first-order temporal logic.

Extended non-clausal resolution rule

$$A\langle u,\ldots,u\rangle,\ B\langle u,\ldots.u\rangle \longmapsto A\langle true\rangle \vee B\langle false\rangle$$

where the occurrences of u in A and B are replaced with true and false, respectively, are all in the scope of the same number of O (Next operator) ‘s and are not in the scope of any other temporal operator in either A or B.

Example : Consider the formulae $O\neg O(\Box p \vee q) \wedge \Diamond\Box p$ and $OO\Box p\ O\Box p$.

(1) $O\neg O(\Box p \vee q) \wedge \Diamond\Box p$

(2) $OO\Box p\ O\Box p$

(3) $[O\neg O(true \vee q) \wedge \Diamond\Box p\ ] \vee [\ OOfalse\ O\Box p\ ]$ $\quad \bot\Box p(2, 1)$

In this case, occurrences of $\Box p$ in the scope of two O’s are replaced with true and false. However, $\Box p$ in $\Diamond\Box p$ and $O\Box p$ cannot be replaced with either true or false because the occurrences of $\Box p$ are in the scope of $\Diamond$ and O temporal operators.

Abadi and Manna [Abadi 1990] introduce six essential modality rules to substitute the O’s versions for other temporal operators.

Modality Rules:

OO rule : $Ou, Ov \longmapsto O(u \wedge v)$

$\Box$ rule : $\Box u \longmapsto u \wedge O\Box u$

$\Diamond$ rule : $\Diamond u \longmapsto u \vee O\Diamond u$

u rule : $u\ \mathsf{u}\ v \longmapsto u \vee (u \wedge O(u\ \mathsf{u}\ v))$; u : Until operator

s rule : $u\ \mathsf{s}\ v \longmapsto \neg v \wedge (u \vee O(u\ \mathsf{s}\ v))$; s : Since operator

In classical logic, all quantifiers can be removed from formulae by application of skolemization rules [Abadi 1990] [Torsun 1995]. However, in first-order logic, quantifiers can be removed only in certain cases. Abadi and Manna propose temporal skolemization rules for first-order temporal logic [Abadi 1990].

The classical skolemization rules

Suppose that A and B are formulae. $Q_1,\ldots Q_n$ are quantifiers, x, $x_1,\ldots,x_n$ are variables, x' is a new variable, and the occurrences of $Q^{\forall}x.B[x]$ and $Q^{\exists}x.B[x]$ under consideration are not in the scope of any quantifiers in A. Classical Skolemization may be described as a two-stage process:

(1) Quantifiers move :
$Q_1x_1\ldots\ldots Q_nx_nA<Q^{\forall}x.B[x]>$ is rewritten to $Q_1x_1\ldots\ldots Q_nx_n\forall x'.A<B[x']>$ and
$Q_1x_1\ldots\ldots Q_nx_nA<Q^{\exists}x.B[x]>$ is rewritten to $Q_1x_1\ldots\ldots Q_nx_n\exists x'.A<B[x']>$

(2) Then Skolem terms replace existentially quantified variables:
$\forall x_1\ldots\ldots\forall x_n\exists x.A[x]$ is rewritten to $\forall x_1\ldots\ldots\forall x_nA[f(x_1,\ldots\ldots x_n)]$,
where f is a new Skolem function symbol. Universal quantifiers stay in place.

4.3.4 Non-clausal resolution for TANDTL logic

As discussed in section 4.2, TANDTL temporal logic does not have any temporal operators. Therefore, modality rules cannot be used for non-clausal resolution in TANDTL. However, a formula u may refer to different time instants as in conventional temporal logic.
Example : Consider the following inconsistent formulae u and v ∏ u. Scope of u is different in two formulae. Therefore sub formula u cannot be replaced with true and false in these formulae.
(1) u
(2) v ∏ u
(3) [(true) $\vee$ (v ∏ false)]$\perp$u(2, 1) ; This is not valid.
However, we can show that
u ∏ v = (u $\wedge$ $\neg$v) ∏ v (F2).

This can be established using the truth table 3.

	T1		T2	Time Zone T1 Follows Time Zone T2	
	u	v	v	$u \prod v$	$(u \wedge \neg v) \prod v$
1	F	F	F	F	F
2	F	F	T	F	F
3	F	T	F	F	F
4	F	T	T	F	F
5	T	F	F	F	F
6	T	F	T	T	T
7	T	T	F	F	F
8	T	T	T	F	F

Table 3 Truth table for $u \prod v = (u \wedge \neg v) \prod v$

Therefore, equation F2 can be used to rearrange the formulae having TAND connective before applying non-clausal resolution. Now in the above example, the formula $(v \prod u)$ can be re-written as $(v \wedge \neg u) \prod u$.

Then applying non-clausal resolution,

(1) u

(2) $(v \wedge \neg u) \prod u$

(3) $\{(\text{false}) \vee [(v \wedge \neg \text{true}) \prod u]\}$ $\quad\quad \perp u(1, 2)$

(4) false

Example : Let us prove that "p before q" and "p after q" are inconsistent.

p before q $= p \prod (\neg p \wedge \neg q) \prod q = [p \prod (\neg p \wedge \neg q)] \prod q$

Applying F2,

p before q $= \{[p \prod (\neg p \wedge \neg q)] \wedge \neg q\} \prod q$

$= \{ (p \wedge \neg q) \prod (\neg p \wedge \neg q) \wedge \neg q\} \prod q$; using distributive law

$= (p \wedge \neg q) \prod ((\neg p \wedge \neg q) \wedge \neg q) \prod q$

$= (p \wedge \neg q) \prod (\neg p \wedge \neg q) \prod q$

p after q $= q \prod (\neg p \wedge \neg q) \prod p$

$= (q \wedge \neg p) \prod (\neg p \wedge \neg q) \prod (p \wedge \neg q)$; applying F2 and distributive law

Then applying non-clausal resolution method,

(1) $(p \wedge \neg q) \prod (\neg p \wedge \neg q) \prod q$

(2) $(q \wedge \neg p) \prod (\neg p \wedge \neg q) \prod p$

(3) $[(\text{false} \wedge \neg q) \prod (\neg p \wedge \neg q) \prod q] \vee [(q \wedge \neg \text{true}) \prod (\neg p \wedge \neg q) \prod p]$;$\perp p(1, 2)$

(4) false

That is the set of formulae { “p before q” and “p after q”} is inconsistent.

Example : Let us consider a simple inventory system which has number of items in stocks. There are two external events in this system : “Sales” and “Purchases”. The event “Sales” decreases stock level while the event “Purchases” increases the stock level of an item. System will function properly if these two events occur sequentially, but inconsistency will occur if two events appear concurrently. This example shows how non-clausal resolution technique can be used to identify the occurrence of inconsistency before the execution. The TANDTL logical statements for the system specifications will be as follows:

S1. Sales $\Rightarrow$ Increase(Item_Quantity, Sales_Quantity)
S2. Purchases $\Rightarrow$ Decrease(Item_Quantity, Purchase_Quantity)
We have to inform the system that both actions “Increase(Item_Quantity, Sales_Quantity)” and “Decrease(Item_Quantity, Purchase_Quantity)” cannot occur concurrently. We can do that by introducing a model rule
S3. $\neg$[Increase(Item_Quantity, Sales_Quantity) $\wedge$ Decrease(Item_Quantity, Purchase_Quantity)]
Let Item_Quantity be IQ, Sales_Quantity be SQ and Purchase_Quantity be PQ. Then the above logical statements can be rewritten as:

S1. $\neg$Sales $\vee$ Increase(IQ, SQ)
S2. $\neg$Purchases $\vee$ Decrease(IQ, PQ)
S3. $\neg$Increase(IQ, SQ) $\vee$ $\neg$Decrease(IQ, PQ)

Applying non-clausal resolution technique:
S4. $\neg$Sales $\vee$ $\neg$Decrease(IQ, PQ) ; $\perp$Increase(IQ, SQ) (1, 3)
S5. $\neg$Purchases $\vee$ $\neg$Increase(IQ, SQ) ; $\perp$Decrease(IQ, SQ) (2, 3)
S6. Increase(IQ, SQ) ; $\perp$Sales (4, 1)
S7. $\neg$Increase(IQ, SQ) ; $\perp$Purchases (2, 5)
S8. F ; $\perp$Increase(IQ, SQ) (6, 7)

That is, the set of logical statements are inconsistent. Therefore we can identify the occurrence of inconsistency and inform the user before the execution of logical statements. However, without the model rule S4, non-clausal resolution technique cannot identify the inconsistency among the above logical statements. Therefore model rule play an important role here.

As shown, non-clausal resolution technique can be used in TANDTL logic to prove the inconsistency of set of temporal formulae (i.e. formulae with $\prod$ connective).

However, TANDTL logic requires a method to remove the quantifiers in order to apply non-clausal resolution for inconsistency proofs. The skolemization rules are employed in first-order logic. These skolemization rules will be reconsidered in the following section and modified so that they can be used in the TANDTL logic. Finally non-clausal resolution technique in first-order logic will be extended to TANDTL logic.

Prenex normal form

Every first-order logical formula has a normal form called prenex normal form. The reason for considering the prenex normal form of a formula is to simplify the proof procedure[Torsun 1995].

Definition [Torsun 1995]

A formula is in prenex form iff either it contains no quantifiers, or it is of the form $Q_1x_1,\ldots..Q_nx_nB$, where B is a formula with no quantifiers (quantifier-free), $x_1,\ldots..,x_n$ (not necessarily distinct) variables, and $Q_i \in \{\forall, \exists\}$, for $i=1,\ldots\ldots,n$. B is sometimes called the matrix.

The following lemma is used to show the every formula is equivalent to a prenex formula.

Lemma [Torsun 1995]

The following formulae are valid.

(a) $\forall x (A \Rightarrow B) \Rightarrow (\forall x A \Rightarrow \forall x B)$.

(b) $\exists x (A \Rightarrow B) \Leftrightarrow (\exists x A \Rightarrow \exists x B)$.

Definition (Bound and free variables)

Let x be a variable. E be an expression (i.e. a formula or a term) of predicate logic, and x occurs in E. The occurrence of x is bound in E if it is within the scope of a quantifier $\forall x$ or $\exists x$.

Theorem [Torsun 1995]

For every formula in first order form, there exists an equivalent prenex form.

Proof.

There is a simple algorithm to transform a formula into an equivalent prenex form.

(1) The connectives $\Rightarrow$ and $\Leftrightarrow$ are eliminated using the following rewriting rules:

$(A \Leftrightarrow B)$ replaced by $(A \Rightarrow B) \wedge (B \Rightarrow A)$

$(A \Rightarrow B)$ replaced by $(\neg A \vee B)$

(2) Bound variables are renamed (if necessary) in such a way that free and bound variables do not share common names. This is required not only for the whole formula, but also for every sub formula.

(3) A quantification whose scope does not contain any occurrence of the quantified variable is suppressed.

(4) All occurrences of the negation are transformed immediately before the atoms, by the use of the following rewriting rules:

$\neg\forall xA \approx \exists x\neg A$.

$\neg\exists xA \approx \forall x\neg A$.

$\neg(A \wedge B) \approx (\neg A \vee \neg B)$.

$\neg(A \vee B) \approx (\neg A \wedge \neg B)$.

$\neg\neg A \approx A$.

(5) All quantifications are transferred in front of the formula, with the help of appropriate rewriting rules. The rules concerning the conjunctions are given below, those concerning the disjunction are obtained by the duality principle.

$(\forall xA \wedge \forall x B) \approx \forall x (A \wedge B)$.

$(\forall xA \wedge B) \approx \forall x (A \wedge B)$ if B does not contain x.

$(A \wedge \forall xB) \approx \forall x (A \wedge B)$ if A does not contain x.

$(\exists xA \wedge B) \approx \exists x (A \wedge B)$ if B does not contain x.

$(A \wedge \exists xB) \approx \exists x (A \wedge B)$ if A does not contain x.

In order to make this set of rules complete, one has to allow the renaming of bound variables.

Example : $\exists xp(x) \wedge \forall xq(x)$ will first be rewritten as $\exists xp(x) \wedge \forall yq(y)$ before applying above rules.

Above theorem can be extended to TANDTL logic by introducing rules (4a) and (5a).

Rule (4a)

$\neg(A \prod B) \approx (A \wedge B) \vee (\neg A \wedge \neg B) \vee (B \prod A)$

Rule (5a)

$(\forall xA \prod \forall x B) \approx \forall x (A \prod B)$.

$(\forall xA \prod B) \approx \forall x (A \prod B)$ if B does not contain x.

$(A \prod \forall xB) \approx \forall x (A \prod B)$ if A does not contain x.

$(\exists xA \prod B) \approx \exists x (A \prod B)$ if B does not contain x.

$(A \prod \exists xB) \approx \exists x (A \prod B)$ if A does not contain x.

As shown above every formula in TANDTL logic can be transformed to prenex normal form. Classical skolemization rules require the formulae in prenex normal form. Therefore classical non-clausal resolution technique can be extended to TANDTL logic. Hence the inconsistency of set of formulae in

TANDTL logic can be determined using extended non-clausal resolution method.

CHAPTER V

EXECUTION OF TEMPORAL LOGIC

This chapter discusses the similarities between logic and the view of the system adopted in this research. Then it shows how the temporal logic TANDTL can be executed. Finally it compares the execution of temporal logic USF with the TANDTL. The purpose of this chapter is to show that the view of a system employed in this research can be implemented using temporal logic TANDTL.

5.1 Logic and the view of a system

Terms in temporal logic are objects while predicates represent relations between objects. Let us have a closer look at syntax 7 in TANDTL. It says that a term can be constructed using another terms, formulae or expressions. Expressions are either terms or formulae. Formulae consist of objects and relations between objects. Since term is an object, according to syntax 7 the following objects can be defined:

1. Atom objects,
2. Sentence objects.

Atoms are relations between objects, such as P(x, y, z) where P is the relation while x, y, z are objects. Therefore P is a relation object. If Q and R are atoms then $\neg$Q, Q $\wedge$ R, Q $\vee$ R, Q $\Rightarrow$ R, Q $\Leftrightarrow$R and Q $\prod$ R are sentences. Now consider the sentence Q $\Rightarrow$ R. It can be considered as a rule as it says that if Q holds then do R. Since syntax 7 allows us to assign this sentence to a object we can have rule objects. On the other way it can be used to create another object encapsulating rules into the object in addition to attribute and methods. Practitioners and researchers have identified the weaknesses in representing business rules in conventional object-oriented systems analysis and design methods. They suggest two solutions;

- One to represent rules as objects [Taylor 1995][Graham 1994][Zaho 1994] and
- Other to encapsulate business rules in to a class[Wu 1995].

These show that the logic can be applied not only with conventional object technologies but also with future object technologies. Finally syntax 7 says that an object can be created as a result of interaction between objects. The view of a system adopted in this research also says that the interaction between objects can either modify an attribute in an object/objects or create new object having new attributes as well as attributes inherited from the parent objects. Hence the view of a system is supported by the TANDTL logic.

Composite rules

Since the logical connectives can be used to connect sentences having the form $Q \Rightarrow R$, there can be a sentence in a form $[(Q1\Rightarrow R1)\wedge(Q2\Rightarrow R2)]$. If $Q1 \Rightarrow R1$ is defined as rule 1 and $Q2 \Rightarrow R2$ as rule 2, our sentence will appear as [Rule 1 $\wedge$ Rule 2]. This is a composite rule. Therefore logic allow us to write composite rules. Of course, as described in the previous section, this composite rule can be considered as a new object.

Composite events/ activities

A sentence in the form $Q \prod R$ is already a composite event/ activity as it commands to do Q and then do R. The events Q and R are grouped together as composite events. As described in the above section, it is possible to create a new object to represent this composite event/ activity.

As explained above, the temporal logic TANDTL is capable of

1. Solving problems appeared in point based and interval based time models,
2. Creating rule objects,
3. Encapsulating rules into an object,
4. Building composite rules and
5. Building composite events/ activities.

Recall that the syntax and semantics of TANDTL is the same as the first - order temporal logic except temporal expressions. Syntax and Semantics of temporal expressions is redefined using TAND logical connective.

5.2 Execution of TANDTL

Logical view of a software requirements specifications

As already discussed, a software requirements specifications states what actions to be invoked when the specified conditions become true. Conditions can be either temporal conditions or non-temporal conditions. Temporal conditions express the chronological order of events/ actions while non-temporal conditions states only the actions to be taken. There are two types of rules: rules which interact with the environment and rules which confine to the internal system. In TANDTL logic all these rules can be expressed by sentences in the form $C \Rightarrow A$ where C is a set of conditions and A is set of actions to be invoked. The set of conditions C can be further divided into two subsets CT and CNT which stand for temporal conditions and non-temporal conditions respectively. Hence sentences $C \Rightarrow A$ can be represented as $(CT \wedge CNT) \Rightarrow A$. Events/ Actions in specifications are relations between objects. Therefore events/ actions can be represented as R(O1, O2,……,On) in logic where R is a relation and O1, O2,……,On are objects.

In addition to the set of rules, a software requirements specification may contain attributes of objects, composite rules and composite events/ activities. Logical representation of them are already discussed.

Execution

Software requirements specification is a set of rules which states actions to be taken when conditions are held. That is

Hold the conditions $\Rightarrow$ Do the actions.

Therefore the following steps are necessary to execute a specification:

1. Check whether the condition is held
2. If the condition is held then activate the actions specified.

Checking the conditions

Checking of non temporal conditions is easy since all the attribute values are available. However, checking of a temporal condition may not be easy. Since TANDTL represents all temporal conditions using TAND logical connective, if we have a methods to check whether A $\prod$ B is true then we can check the truth value of any given temporal condition. Application of temporal database can solve the problem since temporal databases can store records in chronological order and they are capable to answer a query which inquire the chronological order of events A and B. Hence, once an event is executed, we have to create a record with event identifier in the temporal database. The record format would be (E, T) where E is the event identifier and T is the time. Therefore our temporal database will look like (E1, T1), (E2, T2), (E3, T3), …….,(En, Tn).

E1 $\prod$ E2 is true

1. If and only if E1 and E2 are found in the temporal database and
2. T2 comes just after T1.

E1 $\prod$ E2 is false

1. One of E1 or E2 are not found in the temporal database or
2. T2 does not follow T1.

Executing actions

Let us define predicate Exec(A) to represent execution of event A. Execution rule are as follows:

1. Exec(T) = True.
2. Exec(F) = False.
3. Exec(A $\Rightarrow$ B) = Query the temporal database for truth value of A and if it is true execute B. That is Exec(B). If not, do not do anything.
4. Exec(A $\wedge$ B) = Exec(A) $\wedge$ Exec(B).
5. Exec(A $\vee$ B) = Exec(A) $\vee$ Exec(B).
6. Exec(A $\prod$ B) = Exec(A) $\prod$ Exec(B).

Logically, "Exec(A) $\vee$ Exec(B)" is true either A is executed or B is executed. However, in this execution method, execution will be carried out from left to right. Therefore, A will executed first. If A cannot be executed then B will be executed. If both A and B cannot be executed, then the expression "Exec(A) $\vee$ Exec(B)" is false. Since we can write all the temporal knowledge with TAND logical connective, rule 6 provides the basis for execution of temporal events. As mentioned in the previous section, we have to create a record in the temporal database for each execution of events.

<u>Comparison of TANDTL executions with USF executions</u>

This section presents the comparison of TANDTL executions with logic USF [Barringer 1996]. The USF logic employs Until and Since temporal operators to represent all other future and past temporal operators. In USF, specification is considered as a collection of "If past then execute future" rules. That is

hold C in the past $\Rightarrow$ execute B now.

As expressed above, in USF rules, C is always specified using past temporal operators while B is specified using future temporal operators. However, in TANDTL, no past or future temporal operators are available instead all of them are represented using TAND logical connective.

Execution rules of logic USF.

1. Exec(true, m) = true.
2. Exec(false, m) = false.
3. Exec(A $\vee$ B, m) = Exec(A, m) $\vee$ Exec(B, m).
4. Exec(A $\wedge$ B, m) = Exec(A, m) $\wedge$ Exec(B, m).
5. Exec(B Since A, 0) = false.
6. Exec(B Since A, m+1) = Exec(A, m) $\vee$
   [Exec(B, m) $\wedge$ Exec(B Since A, m)].
7. Exec(B Until A, m) = Exec(A, m+1) $\vee$
   [Exec(B, m+1) $\wedge$ Exec(B Since A, m+1)].

Where m is a time point.

Even though the logic USF has logical approach to execute specifications, execution of rules 6 and 7 are not logical but procedural. According to the rule 6, to execute B Since A at time m+1, we have to execute A at time m and only if we can not execute A at time m then execute B at time m together with the B Since A. However, logically both ($A \vee B$) and ($B \vee A$) are same. This shows that the execution of rule 6 is not logical but procedural. Rule 7 has the same weakness. Since TANDTL represents all temporal events using TAND connective and TAND itself possesses the execution order, TANDTL can be executed easily and logically.

The objective of this research is to develop end-user intelligible and executable software requirements language. Now it is clear that a specifications language based on TANDTL is executable but the statements in TANDTL are not understandable to non-technical end-users. The chapter 6 of this thesis explains the language called GSL based on TANDTL and show its intelligibility. Therefore GSL can be considered as the End-user intelligible version of TANDTL. Syntax and semantics of TANDTL are also applied to GSL.

CHAPTER VI

SPECIFICATIONS LANGUAGES

This chapter discusses about present specifications languages and finally introduces new software requirements specification language GSL which is based on temporal logic TANDTL. Since TANDTL is executable, the specifications written in GSL are also executable. Further GSL provides end-user intelligible specifications. Temporal knowledge is represented in GSL using And_Then relation which represents TAND connective in TANDTL. Examples have been provided using the pilot system.

6.1 Conventional specifications languages

This section briefly discusses conventional approaches to software requirements specifications languages and representation of temporal knowledge. The discussion mainly concerns about the end-user intelligibility of the specifications written in these languages and their attempt to represent incomplete relative temporal knowledge. Finally the proposed specifications language GSL (General-purpose Specification Language) will be introduced.

6.1.1 TELOS

Telos[Mylopoulos 1990] is one of latest specifications language developed to represent temporal knowledge in information systems. It employs modified versions of Allen's[Allen 1983] temporal relations (equals, meets, before, overlaps, during, starts and ends). Therefore it inherited the weaknesses of interval based time model. However, Telos was designed to represent incomplete temporal information. For example, infinite time interval is represented as (1986/10/25..*). Let us consider the following Telos specification.

```
TELL TOKEN martin IN paper (at 1986/10..*);
        author
                firstAuthor   : Stanly (at 1986/10..*);
                              : LaSalle (at 1987/1..*);
                              : Wong (before 1987/5)
        title
                : 'The MARTIN system'
END
```

This operation introduces the token martin in the knowledge base. The IN clause makes martin as an instance of the class paper for an unbounded time interval starting October 1986. Similarly, the WITH clause asserts that Stanley is the first author of martin during the interval 1986/10..*, LaSalle is an author during the interval 1987/1,..* while Wong was an author for some time before MAY 1987. Integrity constraints and deductive rules are resented as follows:

```
TELL CLASS Paper IN SimlpeClass WITH
        integrityConstraint
                :$ (∀y/Person)
                   ( y ∈ this.author ⇒ ¬(∃t/Time)y ∈ this.referee[at t]) $
        deductiveRule
                :$ (∀x/Paper) (∀z/Address)
                    ( z ∈ author.address ⇒ z ∈ x.replyAddress) $ (at Alltime)
        END
```

The infinite interval “Alltime” and the special interval “Now” represent the current system time. These show how Telos specifications are rigid. Specifications written in Telos are not always in end-user understandable form. Still end-user has to get some assistance from the system analysts. It is not so easy to use. Users cannot define their own temporal relations. Further, Telos specifications are not executable. Telos dose not support the view of a system adopted in this research.

6.1.2 Templar

The Templar specifications language[Tuzhilin 1995] provides user friendliness to a certain extent, still the end-user needs some assistance from the systems analyst to understand the software requirements specifications. However there is no guarantee that the sentences in Templar specifications are always in end-user intelligible form. It solely depends on the capabilities of the systems analyst. On the other hand, Templar does not employ object-oriented principles directly in its specifications, instead claims that Templar specifications can be converted to object-oriented specifications. Tuzhilin[Tuzhilin 1995] provides a discussion about other specifications languages. Templar specifications language was designed to provide:

1. Powerful and user friendly specifications to system analysts
2. End user understandable specifications
3. Temporal knowledge representation.

Templar uses When clause to represent events, If -Then to represent rules, Then_Do to represent actions. Temporal knowledge is represented by While, Before and After clauses. Sample Templar specification is stated below.

```
When arrives(customer, branch)
While close(branch)
If      has_atm(branch)
Then_Do use_atm(customer, branch)
```

Since first-order predicates are used in Templar specifications, they are not always understandable to end users. Therefore end user has to get some assistance from system analysts to understand the specifications. On the other hand, temporal knowledge is sometimes distributed over clauses in the language and sometimes included in the predicates. Even though Templar provide user defined temporal constructs, it is not easy for users to develop them. For example, to define B Since C one has to write

```
if C then X
if B and Not C and previous X then X
if B and not C and not previous X then not X
if not B and not C then not X.
```

This shows how difficult to define temporal constructs by users. Templar's uses point based temporal operators in predicates and employs while, before and after clauses to represent interval based temporal knowledge. Since Templar is based on first-order temporal logic, it cannot use interval based temporal relations in predicates but as clauses in the language. Templar does not support the view of a system adopted in this research.

6.2 GSL specifications language

This section introduces the new software specifications language proposed in this thesis. GSL is based on TANDTL temporal logic. Therefore it can provide wide range of temporal relations in end user understandable form. The specifications written in GSL is not always grammatically correct but understandable to end users as well as system analysts. Since it is based on executable TANDTL, GSL specifications are also executable.

6.2.1 Structure of GSL

Facts

The basic unit of knowledge representation in GSL is the “Fact”. Facts are common to all knowledge representation methods but they are represented and grouped in different ways. Semantic networks, rules, frames, objects and logics have their unique ways to represent and group facts into a body of knowledge[Torsun 1995]. Facts in GSL have O_1RO_2 (Object1 Relationship Object2) or OR (object Relationship) form. These forms resemble SVO or SV form in elementary English sentences and can be easily transformed to $R(O_1,O_2)$ or R(O) in TANDTL temporal logic. Some researchers[Agusa 1984] model requirements as R(X,Y) where R is the relationship and X,Y are the entities. This approach will inherit problems of the Entity-relationship model. In GSL, a fact by itself can be an object. This new object in conjunction with another object and a relationship can form a new fact and so on**.** Therefore we model a requirement as a series of objects and relationships.

$$\begin{aligned} \text{Fact (F)} = {} & OR && \text{or} \\ & O_1RO_2 && \text{or} \\ & O_1R_1O_2[R_kO_{k+1};\ k=2,n]. \end{aligned}$$

These facts can be transformed to TANDTL atoms very easily. Therefore facts in GSL will be represented in TANDGL as R(O), R(O1, O2) or R1(O1, R2(O2, R3(O3, O4))).

Relationships

In GSL, there are two types of relationships; namely Major and Minor relationships (R_{major} and R_{minor}). All facts must have one major relationship but they may or may not have minor relationships. Minor relationships cannot appear in a fact without a major relationship. We use unary and binary relationships. Tertiary and higher order relationships can be replaced with binary relationships[Torsun 1995]. We replace higher order relationships with one major relationship and one or more minor relationships. The main verb in an elementary English sentence represents a major relationship. That is, a fact does not have a meaning without a major relationship. Therefore a fact (F) can be represented as follows:

Fact (F) = OR_{major} or
$O_1R_{major}O_2$ or
$O_1R_{major}O_2[R_{minor(k)}O_{k+1};k=2,n]$

BNF grammar [Russel 1995] for facts in GSL as follows:

```
Fact  → Fact
        | Object
        | Object Connective Object
        | Object Connective Fact
        | Fact Connective Fact
        | ( Fact )
        | Object Major-relation
        | Object Major-relation Object
        | Object Major-relation Object Minor-relation Object….
        |  Fact GSL-Relation Fact
        | Fact User-Define-Relation Fact
```

User-Define-Relation → If Fact User-Define-Relation Fact Then Fact
Object → Object
| Fact
Connective → **And** | **Or** | **And_Then** | **Not** | **Of**

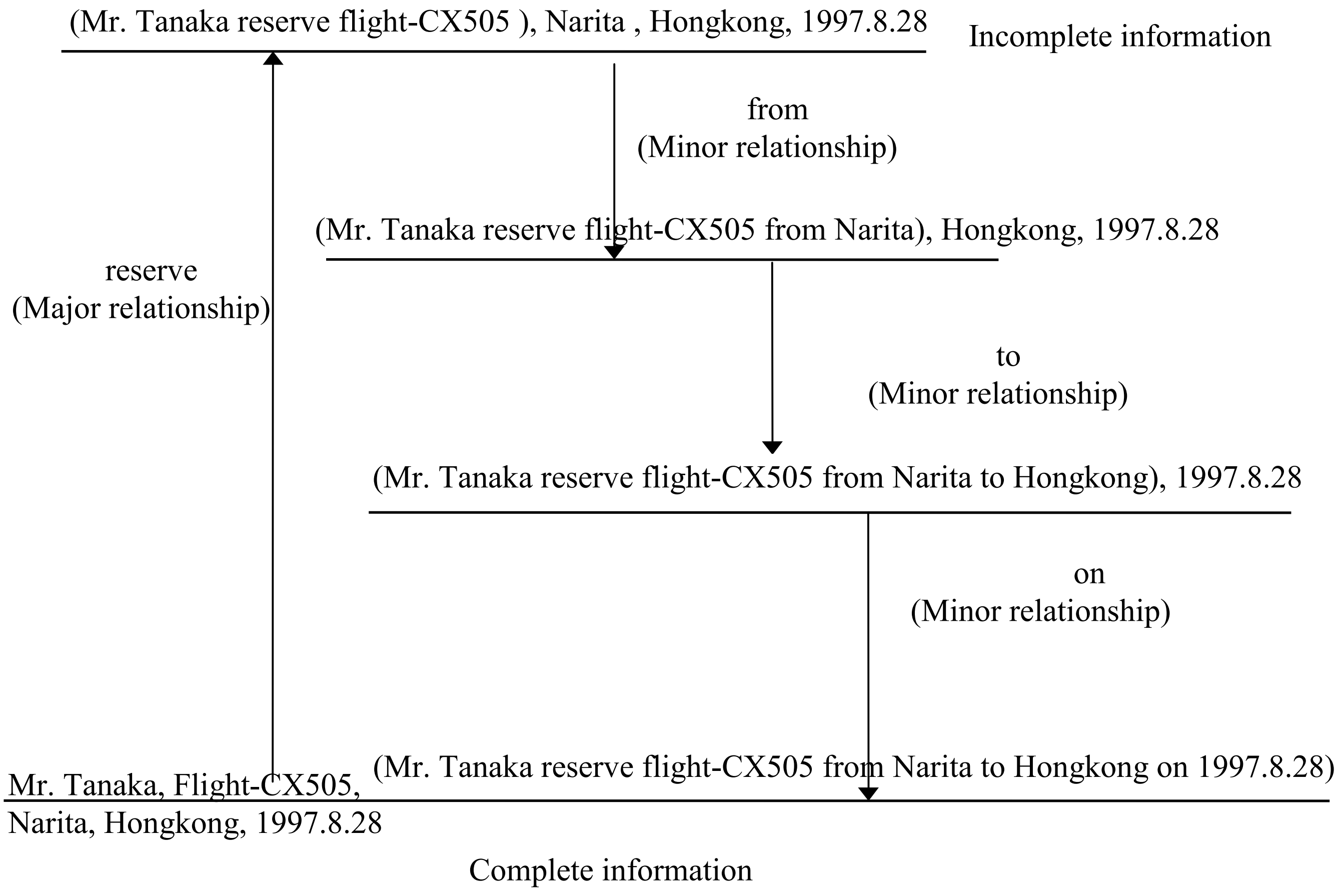
Role of R_{major} in a fact

The relationship R_{major} creates either the most stable or the most unstable fact or the object in a fact. This most stable fact or the object contains complete information while the most unstable fact or the object has incomplete information.

Role of R_{minor} in a fact

The most stable fact or the object created by R_{major} does not require any R_{minor} links with any other object. It is in the most stable state and represents complete information. A fact in the most stable state can have minor relationships with other facts/objects to provide additional information. However, the most unstable fact or the object needs minor relationship/relationships to link with other object/objects to change the most unstable state to the most stable state where complete information is available. Figure 6.1 shows these concepts graphically. Minor relationships are used only to increase the readability and the intelligibility of the specifications while major relationship represents the real actions. All minor relationships can be ignored when a fact is transformed to a TANDTL expression.

Since we consider only elementary English sentences, systems analysts should transform the user statements into elementary sentences that contain only one main verb as suggested by Chen[Chen 1983][Burg 1996]. Present modeling techniques do not represent minor relationships between objects but only major relationships.



(Mr. Tanaka reserve flight-CX505), Narita , Hongkong, 1997.8.28
Incomplete information
from
(Minor relationship)
(Mr. Tanaka reserve flight-CX505 from Narita), Hongkong, 1997.8.28
reserve
(Major relationship)
to
(Minor relationship)
(Mr. Tanaka reserve flight-CX505 from Narita to Hongkong), 1997.8.28
on
(Minor relationship)
Mr. Tanaka, Flight-CX505, Narita, Hongkong, 1997.8.28
(Mr. Tanaka reserve flight-CX505 from Narita to Hongkong on 1997.8.28)
Complete information

Figure 6.1 Role of major and minor relationships in a fact

Example : the fact "A passenger requests for reservation of a flight from Narita to Hongkong " will be represented in Templar[Tuzhilin 1995] as:

Request(Passenger, Flight, Narita, Hongkong)

where "Request" is the major relationship which is the main verb in the statement. As shown in Figure 6.1, this represents incomplete information. However, it will be represented in GSL as:

Passenger request flight from Narita to Hongkong where “request” is the major relationship while “from” and “to” are minor relationships. This form can easily be transformed into the following:

1. request(Passenger, flight, Narita, Hongkong)
2. To(From(request(Passenger, flight), Narita) , Hongkong).

The second form represents the syntax of the fact and it provides complete information. The requirements in the first form are partially complete because it does not give all the information about how terms in a sentence relate to each other. Instead it gives the main verb of statements leaving reader to think the other possible relationships. Therefore representing requirements in the first form, we show fewer information to end-users than what we can get from them. When the number of terms in the sentence is increased, the information loss will also increase. This will lead to a misunderstanding.

The representation method used in GSL will be very useful when additional information needs to be associated with tertiary or higher order relationships.

Example : let us consider the statement “A passenger requests for reservation of a flight from Narita to Hongkong ”. This can be represented by the sentence,

“request(Passenger, flight, Narita, Hongkong)”.

When we want to express the statement “A passenger requests for reservation of a flight from Narita to Hongkong on 1997.8.28”, it is necessary to modify the structure of the sentence as,

“request(Passenger, flight, Narita, Hongkong, 1997.8.28)”.

This requires changes to “request(Passenger, flight, Narita, Hongkong)” wherever it appears. However, in our representation the first statement,

“ *Passenger request flight from Narita to Hongkong*”

can be extended to

" *Passenger request flight from Narita to Hongkong on 1997.8.28*"

to include the additional information without modifying

" *Passenger request flight from Narita to Hongkong*" .

Requirements represented in GSL are not always grammatically correct but intelligible to end-users and also the representation scheme is easy to use by systems analysts. Such requirements representation scheme could not be seen in other requirements specifications languages.

Statements

The statements in GSL have the form

$S = C_iF_i$ or C_iO_i

where C_i = i th clause and F_i = i th fact , O_i = i th object, i=1,n.

BNF grammar for GSL statement is

Statement → Clause Fact Clause

Clause → **When** | **If** | **Then** | **Then_Do** | **Define** | **End_Define**

Connective → **And** | **Or** | **Not** | **And_Then**

GSL-Relation → **Forall** | **Exists** | **As** | **Is_A** | **Of** | **+** | **-** | ***** | **/** | **%**

As shown above User-Define-Relation should always accompany with If-Then rule to the language.

6.2.2 Features of GSL

Logical Connectives

The connectives available in GSL [Wijayarathna 1998] are

And,
Or,
Not and
And_Then.

Truth values **True** and **False** are also available. If F1 and F2 are facts then so are

F1 **And** F2,
F1 **Or** F2 and
F1 **And_Then** F2.

However, the connective **Not** should be attached to a major relationship or fact but not to a minor relationship. Because minor relationships are used only to improve the intelligibility of the specifications. When transforming GSL facts into TANDTL atoms, minor relationships are going to be ignored. Therefore, we define,

if R is a relationship and F is a fact then **Not** R is also a relationship and **Not** F is also a fact.

Highlighted words in GSL specifications are GSL reserve words. Names of objects start with a capital letter. *Italics* words are major relations while simple letter words are minor relations.

Example 3: let us consider the statement

"A passenger does not request for a reservation" .

This is represented in GSL as,

"Passenger **Not** *request* Reservation".

In this representation, the **Not** connective is attached to the major relationship "request". The requirement,

"Tanaka requested a reservation, but the destination is not Colombo", is represented in GSL as

“Tanaka *request* Reservation to **Not** Colombo”.

In this case the minor relationship “to” is not connected to **Not**. This will be represented in Templar[Tuzhilin 1995] as “request(Tanaka, reservation, ¬Colombo)”.

Temporal operators and relations

And_Then (TAND) connective and redefined **And** connective overcome the weaknesses of temporal logic in representing relative temporal knowledge [Wijayarathna 1997]. Therefore in GSL, **And** connective represents concurrent events while **And_Then** ($\prod$) connective represents sequential events. Classical logic with **And_Then** connective is strong enough to represent temporal knowledge. Users can define their own temporal relations or operators using **And_Then** connective. This makes the GSL software requirements specifications language simple but powerful tool in expressing temporal information. GSL establishes a priority order of the connectives and user defined temporal relations/ operators.

Not has the higher priority than **And** and **Or**
And and **Or** have the same priority
And and **Or** have the higher priority than **And_Then**.

All user defined temporal relations can be replaced by **Not**, **And**, **Or** and **And_Then** connectives.

Example : Let us define the temporal relations **Before**, **After**, **While**, **Until**, **At_the_start_of** and **At_the_end_of** using **And_Then** Connective

If p **Before** q
Then p **And_Then not** p **and not** q **And_Then** q

If p **After** q
Then q **And_Then not** q **and not** p **And_Then** p

If p **While** q
Then q **And Not** p **And_Then** p **and** q **And_Then** q **And Not** p

If p **Until** q
Then True **And_Then** q **Or** (p **And_Then** q)

If p **At_the_start_of** q
Then p **And** q **And_Then** q **And Not** p

If p **At_the_end_of** q
Then q **And Not** p **And_Then** p **And** q

As shown in the above examples, users can define their own temporal relation or a temporal operator by adding a simple rule to the language. Therefore users are not restricted to the pre-defined temporal relations. These user defined temporal relations will be transformed to TANDTL sentences as follows:

$(p \text{ Before } q) \Rightarrow p \prod (\neg p \wedge \neg q) \prod q.$
$(p \text{ After } q) \Rightarrow q \prod (\neg q \wedge \neg p) \prod p.$
$(p \text{ While } q) \Rightarrow (q \wedge \neg p) \prod (p \wedge q) \prod (q \wedge \neg p).$
$(p \text{ Until } q) \Rightarrow \text{True} \prod (\neg q \Rightarrow (p \prod q)).$
$(p \text{ At\_the\_start\_of } q) \Rightarrow (p \wedge q) \prod (q \wedge \neg p).$
$(p \text{ At\_the\_end\_of } q) \Rightarrow (\neg p \wedge q) \prod (q \wedge p).$

BNF grammar for **If-Then** statement:
If-Then statement $\rightarrow$ **If** Fact **Then** Fact

Events (Temporal Conditions)

According to the view of a system adopted in this research (section 2.5.2), system starts functioning when an external or an internal event triggers an action(s) under specific condition. Hence GSL specifications should be capable of representing this view of system. **When** clause in GSL does this.

BNF grammar for **When** statement :

When statement → **When** Fact **Then_Do** Fact
| **When** Fact **If** Fact **Then_Do**

This statement can be rewritten as:

WhenEvent
If Condition
Then_Do Action

Using **When** statement, GSL can represent the view of a system adopted in this research. Recall events, conditions and actions are facts.

Example : Now let us consider the requirement number 2 in our pilot system. It says "When the request for reservation is made the consultant should record the passenger details such as name, destination, intended departure date, intended air line and maximum cost". This will be represented in GSL as:

When Consultant *receive* Reservation_request from Passenger
Then_Do Consultant *record* (Name **Of** Passenger **And** Destination **Of** Passenger **And** Departure_date **Of** Passenger **And** Airline **Of** Passenger **And** Max_cost **Of** Passenger) in Request_register

The same requirement will be represented in Templar[Tuzhilin 1995] as:

When*receive*(Consultant, Reservation_request, Passenger)
Then_Do *record*(Consultant, Name, Destination, Departure_date, Airline, Max_cost, Request_register)

Major relations in GSL are represented as predicates in Templar. Names of objects become terms in formulae in Templar. A fact in GSL is the expanded version of formulae in Templar with semantic relations. This improves the intelligibility of the specifications. The event in the above GSL specification is

"Consultant *receive* Reservation_request from Passenger".

The same event appears in Templar as

"*receive*(Consultant, Reservation_request, Passenger)"

In the above example, **Of** relationship represents possession. That is, it stands for the "part-of" relationships in data modeling. The major relationship is "*receive*" while "from" and "in" are minor relationship. The others are objects.

Activities

According to the view of a system adopted, events trigger actions (activities) under specific conditions. As shown above, **When** clause in GSL represents events. The corresponding actions for the events stated in **When** clause are available in **Then_Do** clause. **While** is a temporal relation which can be defined using **And_Then** connective in GSL. However, **While** is a clause in Templar because they employ Event-Action-Condition-Action (EACA) model as the view of a system. If F1 and F2 are facts, then

F1 **While** F2

is also a fact and it is true if and only if F2 is true for some time interval t1 to t2 and F1 is true for some time t3 where $t1 \leq t3 \leq t2$. This fact can appear in **When** clause in GSL. That is an activity can appear in **When** clause in GSL. Therefore it is better to name events in GSL as "temporal conditions". However Templar distinguishes events and actions separately. Hence **When** statement can be represented as:

When	Temporal Condition
If	Non-temporal Condition
Then_Do	Action.

There are no temporal operators available in Non-temporal condition.
Example : the requirement number 11 says that “A passenger can cancel his request for reservation while he is waiting for confirmation without penalty”, This will be states in GSL as:

When Passenger *request* Cancellation **While**
Passenger w*aiting_for* Confirmation
Then_Do Consultant *cancel* Reservation_request **And**
Consultant **Not** *charge* Penalty from Passenger

The corresponding Templar specification is:
When*request*(Passenger, Cancellation)
While *waiting_for*(Passenger, Confirmation)
Then_Do *cancel*(Consultant, Reservation_request)
¬*charge*(Consultant, Penalty, Passenger).

One may not be able to distinguish the difference of appearance of **While** in two specifications. However, “Passenger *request* Cancellation **While** Passenger w*aiting_for* Confirmation” is a single fact in GSL. In the meantime, Templar represents it as two formulae.

The GSL specification can be transformed to TANDTL as :

request(Passenger, Cancellation) **While**
waiting_for(Passenger, Confirmation) ⇒
cancel(Consultant, Reservation_request) ∧
¬*charge*(Consultant, Penalty, Passenger).

Since **While** is a user defined temporal relation in GSL and can be represented using TAND connective in TANDTL, the above formula can be rewritten with TAND connective. For example, suppose that **While** temporal relation is defined as

(p **While** q) $\Rightarrow$ (q $\wedge\neg$p) $\prod$ (p $\wedge$ q) $\prod$ (q $\wedge\neg$p).

Then the above formula can be rewritten as:

{*request*(Passenger, Cancellation)$\wedge\neg$*waiting_for*(Passenger, confirmation)) $\prod$ (*request*(Passenger, Cancellation) $\wedge$
waiting_for(Passenger, Confirmation)) $\prod$(*request*(Passenger, Cancellation) $\wedge$
$\neg$*waiting_for*(Passenger, Confirmation)} $\Rightarrow$
cancel(Consultant, Reservation_request) $\wedge$
$\neg$*charge*(Consultant, Penalty, Passenger).

When executing above TANDTL logical formula, system will check whether the condition in left hand side of "$\Rightarrow$" is true. If it is true, then it will execute both actions *cancel*(Consultant, Reservation_request) and $\neg$*charge*(Consultant, Penalty, Passenger).

In the same way, if F1 and F2 are facts then,

F1 **Before** F2, F1 **After** F2, F1 **Until** F2, F1 **At_the_start_of** F2 and F1 **At_the_end_of** F2 are also facts.

Example : "If a passenger cancels his reservation request before the reservation is confirmed then no penalty for the passenger" is stated in GSL as:

WhenPassenger *request* Cancellation **Before**
Reservation *is* Confirm
Then_Do Consultant **Not** *charge* Penalty from Passenger

Example : “If the passenger does not pay after the reservation is confirmed then the consultant does not issue the ticket” can be represented in GSL as:

WhenPassenger **Not** *pay* Payment **After**
Reservation *is* confirm
Then_Do Consultant **Not** *issue* Ticket.

Example : Requirement 9 in our pilot system says that “If the reservation status is waiting, then consultant asks the passenger whether he is ready to wait. If he is ready to wait then wait for confirmation”. This will be stated in GSL as:

WhenReservation *is* Waiting **And** Passenger *want* Waiting
Then_Do Passenger *wait* **Until** Reservation is Confirm.

When executing the specification, first the condition in **When** clause is evaluated. If it is true, then the actions in **Then_Do** clause will be executed. That is the action “ Passenger *wait”* will be executed until the action “Reservation is Confirm” is executed. Hence, repetitive actions can be stated in GSL requirements specifications.

Example : “If a reservation is confirmed then consultant bill the passenger and create the transaction for accounting” is represented in GSL as:

WhenReservation *is* Confirm
Then_Do Consultant *bill* Passenger **And**
Consultant *create* Transaction for Account.

The **And** connective in TANDTL and GSL represents events occurs simultaneously. Therefore GSL can represent parallel activities with the **And** logical connective.

Example : "Validation is the first step in the transaction recording process." is represented as:

WhenAccount_clerk *receive* Transaction
Then_Do Validation **At_the_start_of** Journalizing **And_Then**
Journalizing

In the above example, the actions in **Then_Do** clause are "Validation" and "Journalizing". These are objects. The view of a system adopted in this research allows creation of new objects with new attributes in addition to the original objects. "Validation" and "Journalizing" are new objects created by some other objects and relations. Suppose that "Journalizing" refers to the following fact.

Clerk *record* Transaction in Journal.

In this fact, Clerk, Transaction and Journal are objects. "*record*" is the major relation and "in" is the minor relation. Then, "Journalizing" is the new object created by Clerk object, Transaction object, Journal object, "*record*" relation and "is" relation. Therefore GSL is capable of representing different level of abstraction. "Journalizing" is a representation of higher level of abstraction while the fact "Clerk *record* Transaction in Journal" represents more detail level of abstraction. In other words, "Journalizing" hides the implementation information. That is a main concept of object orientation. Hence using GSL specification language users can write different level of specifications, more abstract once and more detail ones. That is GSL is scaleable.

<u>Define clause</u>

Define clause is a feature in GSL which is not available in Templar [Tuzhilin 1995] or other specifications languages. It implements the concept of abstraction in object-oriented principles [Sigfried 1996] [Martin 1993]. The view of a system adopted in this research cannot be implemented only by **When** statement described in the previous section. For example, creation of a new object with new attributes in addition to the original attributes cannot be implemented by **When** statement. The role of **Define** clause is to implement that. The syntax of the **Define** clause is either

Define < object | fact > **As** < fact > **End_Define** or
Define < object > **Is_A** < object > **End_Define** or
Define < object > **As** < other construct> **End_Define**

BNF grammar for **Define** statement :

Define statement → **Define** Fact **As** Fact **End_Define**
| **Define** Object **As** Fact **End_Define**
| **Define** Object **As** Object **End_Define**
| **Define** Object **As If -Then** statement

Case 1: **Define** < object > **As** < fact > **End_Define**

In this case, the fact is renamed as an object. This form can be used either to give a name to a fact or to assign a fact/ group of facts to an object.

Example : Let us say that the process of recording a passenger details in the registry is called "passenger_recording" is represented as:

Define Passenger_recording **As** Passenger *record_to* Register **End_Define**

In this example, the fact "Passenger *record_to* Register" is assigned to the object "Passenger_recording". That is, new object "Passenger_recording" is created by objects Passenger and Register and the major relation "*record_to*". This helps systems analysts to use single word "Passenger_recording" instead of "Passenger *record_to* Register" whenever he needs to refer to the recording process. That is, systems analysts will be able to specify requirements in a abstract level and later they can specify the detail implementation of the requirements using GSL specification language. Since "Passenger_recording" is an object by the definition, it can be used as an object in the design and implementation phases, if systems analyst would wish to do so. The corresponding TANDTL statement will be "*passenger_recording* = *record_to(passenger, register*)". Therefore whenever "*passenger_recording*" is called it will replace "record_to(passenger, register)".

Case 2: **Define** < fact > **As** < group of fact > **End_Define**

This is how GSL creates composite facts. It can be either a composite event or a composite activity.

Example : "To record passenger details in the register copy name, destination, departure date, air line, and maximum cost of passenger into register" is represented as :

Define Passenger *record_to* Register **As**
Name **Of** Passenger *copy_to* Register **And**
Destination **Of** Passenger *copy_to* Register **And**
Departure_date **Of** Passenger *copy_to* Register **And**
Airline **Of** Passenger *copy_to* Register **And**
Max_cost **Of** Passenger *copy_to* Register
End_Define

Case 3: **Define** < object > **Is_A** < object > **End_Define**

Example : "A JAL is an air line" is represented as:

Define JAL **Is_A** Airline **End_Define**

This is used to implement the "is-a" relationship in data modeling. Inheritance in GSL is based on the **Is_A** hierarchy. The object JAL in this example is declared as an Airline. Therefore, object JAL can have attributes of the object Airline. The corresponding TANDTL statement will be the "Is_A(JAL, Airline)".

Case 4: **Define** < object > **As** < other construct > **End_Define**
This creates rules or condition objects.

Example : Let us define the rule for penalty charges in rule 14.

```
Define      Penalty10% As
If          Departure_date - Cancel_date > 31 And
            Departure_date - Cancel_date < 60
Then        10 % Payment
End_Define.
```

CHAPTER VII

DISCUSSION

This chapter provides a discussion about TAND connective, temporal logic TANDTL and the specifications language GSL.

When specifying software requirements using GSL specification language, users are allowed to define their own temporal relation. All temporal relations can be defined using **And_Then** connective. For example one can combine Allen's[Allen 1983] (p before q) and (p meets q) together to represent a temporal relation called "on-or-before". The required representation is,

If (p on-or-before q)
Then [p **And_Then** (**Not** p **And Not** q) **And_Then** q] **Or**
[p **And_Then** q].

Yet another example, let us assume that one wants to define a temporal relation "odd-relation" to represent "only p occurs and then only q occurs and then both p and q occur and then only q occurs and then only p occurs". Then,

If (p odd-relation q)
Then (p **And Not** q) **And_Then** **(** **Not** p **And** q) **And_Then**
(p **And** q) **And_Then** (**Not** p **And** q) **And_Then** (p **And Not** q).

The **And_Then** connective can be used to define existing temporal operators in conventional logic. This will increase the expressive power of temporal logic as well as GSL. For example let us consider how to represent the necessary computational steps of a software module to compute the factorial of a given integer "n". The possible steps would be :

1) let z = 1
2) let z = z * n
3) reduce n by 1
4) repeat steps 2 and 3 until n equals 1
5) return z.

Then we can represent this module as,
factorial(n) = (let z=1) ∏ [((let z = z* n) ∏ (reduce n by 1)) until (n=1)] ∏ (return z).

In this representation, **And_Then** is combined with existing temporal operator "until".

As discussed in this thesis, GSL provides end-user understandable software requirements specifications using object-oriented principles. Even though the statements in GSL are not always grammatically correct, end-user will be able to understand the specifications. Systems analysts will enjoy the application of object-oriented technique in conceptual modeling. GSL can be used not only in object-oriented conceptual modeling but also in structured systems analysis and design. Structured systems analysis and designing methods[Hoffer 1996] are still used ubiquitously while the object-oriented methods[Martin 1993] are becoming popular.

Researchers and practitioners[Taylor 1995][Graham 1994][Zhao 1994][Wu 1995] have identified weaknesses in representing business rules in object-oriented systems analysis and designing methods. They suggest two solutions: one to encapsulate business rules into a class and the other to represent rules as rules objects. GSL specifications can be used for both methods since it defines rules as objects and rule objects as attributes of other objects. Therefore GSL is in position to satisfy both present and future conceptual modeling requirements.

Speech interfaces are becoming a reality now[Smith 1994][Zue 1994][Zue 1995]. Systems analysts will also benefit with natural spoken language interfaces in requirements acquisition. They will be able to acquire requirements through natural language spoken interface while they engage in the real working environment. This requires some sort of resemblance between the specifications language and the natural spoken language. GSL can be used with a speech interface since its statements resemble that of English language.

GSL has built-in sequence, condition and repetition constructs. Therefore automated program generation will also be possible with GSL specifications.

Since GSL represents temporal knowledge in facts, reasoning about temporal knowledge is possible. Events in **When** clause will accompany **While**, **Before** and **After** temporal relations, in contrast to **When** clause in Templar[Tuzhilin 1995]. Therefore the implementation in the two models is apparently different.

GSL uses Fact as a modeling construct. Facts in GSL is a grammatically incorrect basic English statements which consist only one main verb and the statements are in active form. Therefore GSL statement can represent three major information components of requirements of information systems; What, Who and When. Since a Fact is in active form, it always starts with the responsible person that is "Who". A Fact itself represent "What" component. Temporal relations in GSL represents "When" component.

CHAPTER VIII

CONCLUSIONS

This chapter concludes this thesis and points out importance of this research. The software requirement specifications language GSL is developed to provide executable, end-user intelligible specifications. Further GSL can represent incomplete relative temporal knowledge with TAND connective. The existing temporal knowledge representation methods need complete time information about occurrences of events which cannot be found in information systems. However, new logical connective TAND does not require complete time information. It can represent incomplete relative temporal knowledge without any difficulty. The executable temporal knowledge TANDTL is developed using TAND connective to execute GSL specifications. Therefore TANDTL has only TAND connective in addition to standard logical connectives. This simplifies the both logic and the execution process. Since GSL is based on TANDTL temporal logic, GSL specifications are executable. Hence, GSL can solve the problems with accounting systems computerization.

The software requirements specifications language GSL (General-purpose Specification Language) is designed to satisfy following requirements:

1. End-user Intelligible, executable software requirements specifications
2. Application of object-oriented principles in requirements specifications
3. Representation of relative temporal knowledge.

Requirements specifications written in GSL are easily understandable to end-users than specifications written in other specifications languages. Also systems analysts will be able to use object-oriented concepts in conceptual modeling. Programs which satisfy the requirements specifications can be generated quickly from the executable software requirements specifications.

GSL is based on temporal logic TANDGL. The temporal logic TANDGL is developed removing all existing temporal operators and relations and using TAND connective. Therefore TANDTL consists of TAND logical connective in addition to standard logical connective.

The logical connective TAND is proposed to represent temporal knowledge because the existing temporal operators and temporal relations require complete time information to represent temporal knowledge. However, introduction of TAND connective simplifies the representation scheme and it is capable of representing incomplete relative temporal information which can be found in any given information system.

Since TANDTL is executable and GSL is based on it, the GSL specifications is also executable. Again TANDTL simplifies the execution process because it applies only TAND connective to represent temporal knowledge. Therefore execution of temporal actions can be done by executing TAND.

Finally, this thesis proposes a new logical connective TAND to represent incomplete relative temporal knowledge, a new temporal logic TANDTL using TAND connective and a new software specifications language GSL based on TANDTL. GSL specifications are executable and intelligible to end-users.

When writing GSL specifications for example system, it was found that the translation of requirements from source document to GSL specifications is easy. However, in some cases, it was required to identify events in order to write the specifications since the source document itself does not provide them. Therefore a source document written in natural language may require some modifications before translating to GSL specifications.

ACKNOWLEDGEMENT

I wish to express my heartiest gratitude to my academic supervisor Professor M. Maekawa for his guidance, encouragement and valuable advice given to me throughout the research project and making comments which enabled me to improve both the content and presentation of the thesis.

My sincere thanks are due to Research Assistants Mr. Y. Kawata, Mr. Soshi Masakazu and all members of Maekawa laboratory for their kind help and assistance given to me during the course of project.

I would like to extend my sincere gratitude towards Mr. Kinya Iwata and family for their numerous help and encouragement given to me during my course of studies especially on arrival to Japan.

I would also like to convey a special word of thanks to Randeniya and Silva family for their encouragement and assistance given to me in order to overcome difficult situations.

Finally I would like to thank my wife Dilrukshi for her utmost support given to me during my course of studies.

BIBLIOGRAPHY

[Abadi 1990] M. Abadi and Z. Manna, “Nonclausal Deduction in First-Order Temporal Logic”, Journal of the Association for Computing Machinery, **37**, (2), 279-317 (1990)

[Allen 1983] J .F. Allen, “Maintaining Knowledge about Temporal Intervals”, Communication of the ACM, **26**(11), 832- 843 (1983)

[Allen 1984] J. F. Allen, “Towards a General Theory of Action and Time”, Artificial Intelligence, **23**, 123- 154 (1984)

[Barringer 1996] H. Barringer, M. Fisher, D. Gabbay, R. Owens and M. Reynolds, The Imperative Future: Principles of Executable Temporal Logic, John Wiley & Sons Inc., 1996

[Bassiouni 1994] M. A. Bassiouni, A. Mukherjee, and M. J. Llewellyn, “Design and Implementation of Extended Boolean and Comparison Operators for Time-Oriented Query Languages”, The Computer Journal, 37(7), 576-587 (1994)

[Bekke 1992] J. H. T. Bekke, Semantic Data Modeling, Prentice Hall International (UK) Ltd., 1992

[Borgida 1985] A. Borgida, S. Greenspan and J. Mylopoulos, “Knowledge Representation as the basis for Requirements Specifications”, IEEE Computer, 82-91 (1985)

[Chen 1976] P. P.-S. Chen, “The Entity-Relationship Model - Toward a Unified View of Data”, ACM Transactions on Database Systems, **1**, (1), 9-36 (1976)

[Davis 1990] J. R. Davis, C. W. Alderman, L. A. Robinson, “Accounting Information Systems : A Cycle Approach”, Third Edition, John Wiley & Sons, 1990

[Eisen 1994] P. J. Eisen, Accounting, Third Edition, Barron's Educational Series, Inc., (1994)

[Fisher 1995] M. Fisher and R. Owens (eds.), "Executable Modal and Temporal Logics", Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), **897**, Springer-Verlag, 1995.

[Fraser 1991] M. D. Fraser, K. Kumar and V.K. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap", IEEE Transactions on Software Engineering, **17**, (5), 454-466 (1991)

[Galton 1990] A. Galton, "A Critical Examination of Allen's Theory of Action and Time", Artificial Intelligence, **42**, 159- 188 (1990)

[Graham 1994] I. Graham, "The SOMA Method: Adding Rules to Classes", Object-Oriented Methods, Addison-Wesley, 1994

[Hall 1990] A. Hall, "Seven Myths of Formal Methods", IEEE Software, 11-19 (1990)

[Hoffer 1996] J. A. Hoffer, J. F. George and J. S. Valacich, Modern Systems Analysis and Design, The Benjamin / Cummings Publishing Company, Inc., 1996

[Hsia 1993] P. Hsia, A. Davis and D. Kung, "Status Report: Requirements Engineering", IEEE Software, 75-79 (1993)

[Jungclaus 1996] R. Jungclaus, G. Saak, T. Hartmann and C. Sernadas, "TROLL - A Language for Object-Oriented Specification of Information systems", ACM Transactions on Information systems, **14**, (2), 175-211 (1996)

[Kripke 1959] S. Kripke, "A completeness of theorem in model logic", Journal of Symbolic Logic, **24**, 1 -14 (1959)

[Kroger 1987] F. Kroger, Temporal Logic of Programs, Vol. 8, Theoretical Computer Science Series, Springer-Verlag, 1987

[Lamport 1994] L. Lamport, “The Temporal Logic of Actions”, ACM Transactions on Programming Languages and Systems, **16**, (3), 872-923 (1994)

[Manna 1995] Z. Manna and A. Pnueli, Temporal Verification of Reactive Systems (Safety), Springer-Verlag New York, Inc., 1995

[Martin 1993] J. Martin, Principles of Object - Oriented Analysis and Design, Prentice-Hall International, Inc., 1993

[McDermott 1982] D. McDermott, “A Temporal Logic for Reasoning about Processes and Plans”, Cognitive Science, **6**, 101-155 (1982)

[Mylopoulos 1990] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, “Telos: Representing Knowledge about Information systems”, ACM Transactions on Information systems, **8**, (4), 325-362 (1990)

[Needles 1993] B. E. Needles, H. R. Anderson and J. C. Caldwell, Principles of Accounting, Fifth Edition, Houghton Mifflin Company, (1993)

[Peter 1992] V. B. Peter, “Reasoning about qualitative temporal information”, Artificial Intelligence, **58**, 297-326 (1992)

[Russell 1995] S. Russell and P. Norvig, Artificial Intelligence : A modern Approach, Prentice-Hall International, Inc., 1995

[Sigfried 1996] S. Sigfried, Understanding Object-Oriented Software Engineering, IEEE Press, 1996

[Smith 1994] R. W. Smith, “Spoken Variable Initiative Dialog: An Adaptable Natural-Language Interfaces”, IEEE Expert, 45-49 (1994)

[Taylor 1995] D. Taylor, "Business Objects: Building Intelligent Objects", Object Magazine, July - August, 18-19 (1995)

[Torsun 1995] I.S. Torsun, Foundation of Intelligent Knowledge-Based Systems, Academic Press, 1995

[Tuzhilin 1995] A. Tuzhilin, "Templar: A Knowledge-Based Language for Software Specifications Using Temporal Logic", ACM Transactions on Information systems, **13**, (3), 269-304 (1995)

[Ullman 1982] J. D. Ullman, Principles of Database Systems, Second Edition, Computer Science Press, Inc., 1982

[Vessey 1994] I. Vessey and S.A. Conger, "Requirements Specification: Learning object, Process, and Data Methodologies", Communications of the ACM, **37**, (5), 102-113 (1994)

[Vilain 1982] M. Vilain and H. A. Kautz, "Constraint propagation algorithms for temporal reasoning", Proceedings AAAI-86, Philadelphia, PA, 132-144 (1986)

[Wijayarathna 1997] P. G. Wijayarathna, Y. Kawata, A. Santosa, K. Isogai and M. Maekawa, "Representing Relative Temporal Knowledge with the TAND Connective", Proceedings, Eight Ireland Conference on Artificial Intelligence, Vol.2 , 80-87 (1997)

[Wijayarathna 1998] P. G. Wijayarathna, Y. Kawata, A. Santosa, K. Isogai and M. Maekawa, "GSL : A Requirements Specification Language for End-User Intelligibility", Software : Practice & Experience, **28**, (13) 1998.

[Wilkinson 1993] J. W. Wilkinson, Accounting Information systems: Essential Concepts and Applications, Second Edition, John Wiley & Sons, Inc., 1993

[Wu 1995] X. Wu, S. Ramakrishnan and H. Schmidt, "Knowledge Objects", Informatica, **19**, (4), 1-15 (1995)

[Zhao 1995] L. Zhao and E. Foster, "ROO: rules and object-orientation", TOOLs Pacific '94 Technology of Object-Oriented Languages and Systems, 31-44 (1994)

APPENDIX (A)

AN EXAMPLE SYSTEM : CASE STUDY

This appendix shows an application of GSL specification language for a real problem. This assignment case appears in appendix A in “Accounting Information systems: Essential Concepts and Applications”[Wilkinson 1993].

Datacruncher Officer Equipment, Inc.

Statement

Datacruncher Office Equipment, Inc. of Dallas, Texas, is a manufacturer of varied machines and devices for the modern office. Among its products are desk calculators, terminals, printer units, key-to-tape units, micro film reader, word processing systems, time stamping machines, and addressing devices. The firm distribute its products nationwide through 150 franchised dealers who also handle the product of competitors. In addition, the firm sells direct to large and medium-size business firms and other organizations having substantial data processing requirements. The firm also provides services to its customers. About 600 customers receive statements at the end of a typical month of sales or service.

Datacruncher was started in the late 1970s. The founders were four employees - two sales persons and two engineers - from a long established office machines manufacturer. They foresaw the growing importance of the office in the modern firm. Their vision has been amply rewarded, since their firm has enjoyed an explosive growth. Of course, vision alone was not sufficient to generate this growth. A sound knowledge of the office equipment market and skillful designed products were the essential ingredients.

The firm's growth is reflected by several measures. Sales have reached $70 million during this year just ended; this amount represents a 20 percent increase over last year's sales. The number of managers and employees has climbed to almost 1100 as of this year end. Approximately 500 suppliers provide materials and parts for the 120 products that the firm manufactures. The physical facilities consist of the home office building and production plant, located just off an express parkway in Dallas, plus three regional sites in San Francisco, St. Louis, and Philadelphia. Each regional site contains a sales office and a warehouse. A finished-goods warehouse is attached to the plant in Dallas.

Besides growth in sales the firm's founders have emphasized the need to increase the firm's share of the office equipment market and the return on total assets. To achieve these objectives, they have stressed aggressive salesmanship, new-product development, prompt deliveries of ordered products, minimized production and inventory cost, and prudent cash management.

Organization

Datacruncher is organized as a corporation and has 2500 stockholders. The board of directors consists of the four founders plus four outside directors. The four founders occupy top positions in the firm: Bill Dixon is the president, Bert Sanders is the vice president of marketing, Judy Hollis is the vice president of engineering, and Jim Marshall is the vice president of production. The first two were formerly salespersons; the last two were engineers. Other high-level managers include Harry Myler, vice president of administration; Charles Dauten, vice president of finance; Barbara Fulton, controller; and Tim Baker, director of information systems.

Financial status

The income statement for the last two years (as shown in Figure A.1) reflect the sales growth mentioned earlier. They also show, however, that net income has grown less rapidly than net sales. In fact, net income for this year has declined from last year's net income. There are indications that this decline stems from two factors: (1) rising costs in production, inventory, and other areas; and (2) necessary reductions in the prices of certain products to combat the new products of competitors.

The balance sheets for the last two years, (shown in Figure A.2) indicate that the firm's financial position is basically sound. However, there are certain adverse signs, such as a shrinking cash balance.

Procedures

Four broad activities at the operational level can be identified as the revenue cycle, the production cycle, the expenditure cycle, and inventory management. Other activities include engineering design, market research, personnel and payroll, cash management, and general ledger accounting. Processing of transactions pertaining to the first four preceding activities, plus payroll and general ledger accounting, is aided by two computers. One of these computers is located within the accounting function, the other within the production function. Magnetic disks are employed for on-line storage of files. The following sections describe briefly the current processing, including key documents, outputs, and files. (In addition, Figure A.3 contains measures of activity relating to the following procedures.)

Revenue Cycle

Salespersons periodically visit the dealers and prospective business firms and other organizations in their sales regions. As they obtain orders, they may mail or phone in the orders to their regional sales offices. Each sales office then records the orders on a register and prepares formal sales orders in quadruplicate. The original of each order is mailed to the customer, whereas the last copy is filed by customer name. At the end of each day the batch of orders (consisting of the middle two copies of all orders prepared that day) is mailed to the home office.

When received in the sales order department, the orders are reviewed for completeness and accuracy by sales order clerks and numbers are assigned. The orders are then forwarded to the credit department for credit check. When credit is approved for the amount of the orders, one copy of each order is sent to the inventory control department and the other copy to the billing department.

By reference to computer printouts of product status, inventory control clerks determine whether or not sufficient inventory is available to fill each order. If sufficient inventory is available at the warehouse in the region where the customer resides, one copy of the order is mailed there. If sufficient inventory is not available in the regional warehouse but is available in the main finished-goods warehouse in Dallas, the copy of the order is routed there instead. In either case the goods are picked and readied for shipment, based on the order. A shipping report and bill of landing are prepared and the order is shipped. A copy of the shipping report is enclosed with the shipment as a packing slip, and another copy is returned to the billing department. If sufficient inventory is not available to fill the order in its entirety, the inventory control clerk prepares a backorder, which he or she routes to the production planning and control department.

On the receipt of a shipping report, a billing clerk pulls the department's copy of the sales order from a file, verifies that the product numbers and quantities match, and notes the shipping date and prices on the order copy. Next he or she sends the order copy (together with the other orders processed that day) to the data preparation section in the accounting data processing department. The orders then are keyed onto magnetic tapes, edited, sorted, and processed against the accounts receivable and finished-goods inventory (product) master files. Sales invoices are generated as outputs from this processing, together with an open sales invoice file on magnetic disk. Two copies of each sales invoice are mailed to a customer, two other copies are sent to the sales order department and the appropriate regional sales office, and a fifth copy is filed alphabetically.

All cash receipts from customers are received in the mailroom at the home office. There they are opened and listed on a special form. Then the cheques are routed to the cashier, together with a copy of the list. The cashier prepares a bank deposit slip in duplicate, endorse the cheques, and delivers the deposit to the bank the next morning. A copy of the deposit slip is returned to a file in her office. Another copy of the listed receipts is sent to the credit department, where a clerk enters the customer numbers that corresponds to the listed names and address. The clerk forwards the list to the data preparation section, which keys the receipts data onto magnetic tape. The transaction data are then sorted and processed against the accounts receivable master file once each day. At the end of the month the accounts receivable master file is processed to produce an accounts receivable aging schedule, which is sent to the credit manager, and statements, which are mailed to customers.

Production Cycle

Products are manufactured either for inventory or to fill backorders. The overall production level generally is based on a sales forecast made by the marketing function. However, backorders occur because of out-of-stock conditions, and must be fitted into the schedule. In fact, backorders are given priority in order to pacify unhappy customers.

Production operations are triggered when the production planning and control department receives production authorizations or backorders. The production authorizations are prepared by comparing forecasted sales levels with current levels of finished-goods-inventory on hand and are issued jointly by the production superintended and the inventory control manager. Backorders are prepared as described earlier, on the basis of orders that cannot be filled. The production planning and control department then obtain the bills of material from the engineering function and explores the production requirements to determine materials and parts requirements. With the materials and parts requirements in hand, a production planning clerk checks a computer printout of materials and parts inventory on hand. If the materials and parts on hand are adequate for a particular product, the clerk schedules a production run (based on available labor and machines). As each scheduled date nears, the clerk sends the affected production authorizations and backorders to the data preparation section of the production data processing department. There the production requirements data are keyed onto magnetic tape, sorted by product number, and processed to produce numbered production orders, materials requisitions, and move ticket. Files used in this processing run (all on magnetic disk) are the bill-of-materials file, the operations list file, the open production order file, and the work-in-process inventory master file.

Copies of the materials requisitions are sent to the materials storeroom, which then delivers materials and parts to the designated production departments. Copies of the production orders and the move tickets are sent to the first production department involved in the manufacturing process (usually the fabrication department). Copies of production orders are also sent to the cost accounting department, and copies of materials requisitions are kept in the data processing department for inventory processing. As work is completed on an order in a department, a move ticket is returned to the production planning and control department. At the end of each day, all returned move tickets are batched and forwarded to the data preparation section. There the move ticket data are keyed onto a magnetic tape, sorted by production order number, and processed to produce a daily production status report. The open production order file is updated during the processing.

In separate daily processing steps, the materials requisitions are batched, keyed onto a magnetic tape, sorted by material-part number, and processed to update the raw-materials inventory master file. Then the materials requisitions are resorted by production order number and processed (together with labor job-time tickets forwarded from work centers and sorted in a like manner on a separate magnetic tape) to update the work-in-process inventory master file.

When a production order has progresses through the fabrication and assembly departments, the units of completed products are inspected. Those units that pass inspection are released to the finished-goods warehouse, and copies of the order release are sent to the production planning and control department and the cost accounting department. From the central warehouse the finished products are shipped, via stock transfer notices, to the three remote warehouses as needed to replenish stocks. The production planning and control department records the completion and then sends the releases to the data preparation section. There they are keyed onto a magnetic tape, sorted by product number, and processed against the finished-goods and work-in-process inventory master file, as well as the open production orders file. A completed production orders report is also printed; it includes the costs charged to each order.

When the materials and parts needed to manufacture particular products are not available, the production planning and control clerk prepares purchase requisitions. These requisitions are sent to the purchasing department.

Expenditure Cycle

A wide variety of expenditures, ranging from utilities to insurance, are necessary. Expenditures for raw materials and parts, as well as subassemblies, are particularly significant, since the products manufactured by the firm require a high level of precision. Thus, the procedure pertaining to the purchases of such items and the disbursements for them is another of the critical transaction cycles within Datacruncher.

Purchases are initiated by either production planning and control clerks or inventory control clerks. The former clerks issue purchase requisitions when they note that materials and parts are not adequate for upcoming production runs, whereas the later clerks issue similar documents when their experience suggests that the on-hand quantities of particular items have declined to reorderable levels. On the basis of these purchase requisitions, buyers in the purchasing department select suppliers who are known to be reliable and enter their codes on the requisitions, together with acceptable price for the items to be ordered. The requisitions are then forwarded to the data preparation section in the production data processing department. There they are keyed onto magnetic tape, sorted by supplier number, and processed to produce purchase orders. During subsequent runs, the raw-materials inventory master file and the open purchase order file are updated. The purchase orders are then signed by the purchasing manager and mailed to the suppliers. Copies of the purchase orders are forwarded to the receiving department and the accounts payable department, and a fourth copy is filed by supplier name in the purchasing department.

When ordered materials and parts arrive at the receiving dock, receiving clerks pull the purchase order copies from their file. Then they count or weigh the items and prepare receiving reports. The items are next transferred to the materials storeroom and the initialed copies of the receiving reports are sent on to the accounts payable department and filed. Another copy of receiving report is sent to the data preparation section of the data processing department, and a third copy is filed numerically in the receiving department. In the data preparation section, the receiving reports are keyed onto a magnetic tape, sorted by material-part number, processed to update the raw-materials inventory master file, resorted by purchase order number, and processed to update the open purchase order file.

When suppliers' invoices arrive in the accounts payable department, clerk pull the receiving reports and purchase orders from the file and compare the documents. After completing their vouching of the invoices, they prepare disbursement vouchers, record them in a voucher register, and file all the documents together by payment due date.

Each day other clerks pull the vouchers due for payment that day and send them to the data processing department. There the payment data are keyed onto magnetic tape, sorted by supplier account number, and processed to produce cheque vouchers and a cheque register. The accounts payable master file is also updated during this run; in effect, each affected supplier's account is credited to reflect the obligation and debited to reflect the payment.

Inventory Management

Three inventory files are maintained by the firm. The raw-materials inventory master file is updated to reflect orders for materials and parts, as well as receipts from suppliers and issues into production. The finished-goods inventory master file is updated to reflect the newly manufactured products and the sales of products to customers. The work-in-process inventory master file is updated to reflect the start in production of each production order, the issues of materials into production, the charges of labor (from job-time records) into production, the application of over-head to production, and the completion of production.

Problems

A number of specific problems have become apparent. Some of these problems relate to the procedures described earlier, whereas other problems arise from weaknesses in the organizational structure and in financial planning. Many of these problems stem from the fact that the founders have focused on selling and engineering. They have not given as much attention to the areas of accounting, finance, production, and inventory management. Most of the problems also arise from rapid growth in sales.

Some of the significant problems, in addition to those noted earlier, should be mentioned. Interest costs are relatively high, as are costs in production and distribution. Backorders are fairly numerous, even though inventory levels have been rising. Promised delivery dates on customers' orders are often missed, even though lead times of two weeks or more often are allowed. Processing backlogs are sustained in several of the accounting department. These backlogs lead to a variety of ill effects; for instance, purchase discounts are frequently lost and numerous errors are introduced into the transaction data. The percentage of products that do not pass inspection is rather high, perhaps at least in part because of fairly obsolete production equipment and a high labor turnover. Production schedules are difficult to keep up to date, and production jobs often fall behind their schedules. In fact, production clerks keep extremely busy "pushing" jobs, monitoring their progress, and answering phone calls from concerned customers and salespersons. Also, production rates tend to fluctuate, so that production employees are idled at other times and required to work overtime at other times. This problem stem in part from rush backorders; however, it also arises from sales forecasts that prove to be quite inaccurate and from planning procedures that are relatively weak. For instance, the budget process is fairly rudimentary. Budgets are not tied to carefully established cost standards, are not developed in detailed formats, and are not revised to reflect changed conditions. Finally, the reports provided to managers are rather inadequate; most are of the status variety, such as the weekly materials-and-parts status report, the daily product-status report, and the monthly report of budgeted costs versus actual costs.

Initiation of Systems Development

These problems, and their effects on the firm's financial status, have been of concern to the founders for some time. Their view is that at least some of the problems are aggravated by an inadequate accounting information system. Recently, in fact, they created the position called director of information systems and hired Tim Baker, because they strongly felt that corrective measures were necessary. Perhaps, they thought, he could harmonize and update an accounting information system that is rather uncoordinated and somewhat obsolete at present.

Highlighted words represent GSL clauses and relations, *italic words* are used for major relations. Words started with a capital letter indicate objects while the words started with a simple letter represent minor relations.

GSL specifications for Revenue Cycle

When Salesperson *obtain* Order
Then_Do Salesperson *mail* **Or** *Phone* Order to RegionalSalesOffice

When RegionalSalesOffice *receive* Order
Then_Do RegionalSalesOffice *record* Order on Register **And_Then** RegionalSalesOffice *prepare* OriginalCopyOfOrder **And** SecondCopyOfOrder **And** ThirdCopyOfOrder **And** LastCopyOfOrder **And_Then** RegionalSalesOffice *mail* OriginalCopyOfOrder to Customer **And_Then** RegionalSalesOffice *store* LastCopyOfOrder

```
            by CustomerName

Define      OriginalCopyOfOrder Is_A Order End_Define
Define      SecondCopyOfOrder Is_A Order End_Define
Define      ThirdCopyOfOrder Is_A Order End_Define
Define      LastCopyOfOrder Is_A Order End_Define
Define      Backorder Is_A Order End_Define

When        RegionalSalesOffice end Day
Then_Do     RegionalSalesOffice mail Order to HomeOffice

When        SalesOrderDepartment Of HomeOffice receive Order
            from RegionalSalesOffice
Then_Do     SalesOrderClerk Of SalesOrderDepartment review Order
            for Completeness And Accuracy

WhenS       alesOrderDepartment review Order  After
            SalesOrderDepartment Of HomeOffice receive Order
            from RegionalSalesOffice
If          Order is Complete And Accurate
Then_Do     SalesOrderClerk assign Number to Order And_Then
            SalesOrderClerk send Order to CreditDepartment
            for CreditCheck

When        CreditDepartment approve Credit for amount Of Order
            After CreditDepartment receive Order
            from SalesOrderDepartment
Then_Do     CreditDepartment send SecondCopyOfOrder to
            InventoryControlDepartment And_Then CreditDepartment
            send ThirdCopyOfOrder to BillingDepartment

When        InventoryControlClerk Of InventoryControlDepartment
            receive SecondCopyOfOrder from CreditDepartment
```

```
Then_Do     InventoryControlClerk perform checkForSufficientInventory

Define      Warehouse at Area Of Customer As Warehouse End_Define

Define      MainFinishedGoodsWarehouse at Dallas As MainWarehouse
End_Define

Define      CheckForSufficientInventory As
            InventoryControlClerk checkfor SufficientInventory
            in Warehouse  Or MainWarehouse
            byreferringto ProductStatusReports After
            InventoryControlClerk Of InventoryControlDepartment
            receive SecondCopyOfOrder from CreditDepartment
End_Define

When        InventoryControlClerk found SufficientInventory
            at Warehouse While CheckForSufficientInventory
Then_Do     InventoryControlClerk mail Order to Warehouse


When        InventoryControlClerk Not found SufficientInventory
            at Warehouse While CheckForSufficientInventory
Then_Do     InventoryControlClerk mail Order to MainWarehouse

When        CheckForSufficientInventory
If          OutOfStock
Then_Do     InventoryControlClerk prepare Backorder And_Then
            InventoryControlClerk send Order to
            ProductionPlanningAndControlDepartment

Define      OutOfStock As
            Quantity = 0
End_Define
```

When Warehouse **Or** MainWarehouse *receive* Order
from InventoryControlDepartment
Then_Do Warehouse **Or** MainWarehouse *prepare* Shipment
for Order **And_Then**
Warehouse **Or** MainWarehouse *prepare*
ShippingReport **And** BillOfLanding
And_Then Warehouse **Or** MainWarehouse *enclose*
ShippingReport with Shipment **As** PackingSlip
And_Then Warehouse **Or**
MainWarehouse *enclose* BillOfLanding with Shipment
And_Then Warehouse **Or** MainWarehouse
ship Shipment **And_Then**
Warehouse **Or** MainWarehouse *send*
ShippingReport to BillingDepartment

When BillingClerk **Of** BillingDepartment *receive*
ShippingReport
Then_Do BillingClerk *retrieve* LastCopyOfOrder **And_Then**
BillingClerk *verify* ProductNumber **And** Quantity
And_Then BillingClerk *write* ShippingDate **Of**
ShippingReport on LastCopyOfOrder **And_Then**
BillingClerk *write* Price **Of** ShippingReport on
LastCopyOfOrder **And_Then**
BillingClerk *send* LastCopyOfOrder to
DataPreparationSection **Of**
AccountingDataProcessingDepartment

When AccountingDataProcessingDepartment *receive*
LastCopyOfOrder from BillingDepartment
Then_Do AccountingDataProcessingDepartment *retrieve*
LastCopyOfOrder **And_Then**

```
            AccountingDataProcessingDepartment print
            FirstCopyOfSalesInvoice And
            SecondCopyOfSalesInvoice And
            ThirdCopyOfSalesInvoice And
            FourthCopyOfSalesInvoice And
            FifthCopyOfSalesInvoice And_Then
            AccountingDataProcessingDepartment send
            FirstCopyOfSalesInvoice And
            SecondCopyOfSalesInvoice to Customer And_Then
            AccountingDataProcessingDepartment send
            ThirdCopyOfSalesInvoice And
            FourthCopyOfSalesInvoice to
            SalesOrderDepartment And_Then
            AccountingDataProcessingDepartment store
            FifthCopyOfSalesInvoice in AlphabeticalOrder

Define      FirstCopyOfSalesInvoice And SecondCopyOfSalesInvoice
            And ThirdCopyOfSalesInvoice And
            FourthCopyOfSalesInvoice And FifthCopyOfSalesInvoice
            Is_A SalesInvoice
End_Define

When        Mailroom Of HomeOffice receive CashReceipt
Then_Do     Mailroom list CashReceipt on SpecialForm And_Then
            Mailroom send Cheque And SpecialForm to Cashier

When        Cashier receive Cheque And SpecialForm from Mailroom
Then_Do     Cashier prepare FirstCopyOfBankDepositSlip And
            SecondCopyOfBankDepositSlip And_Then
            Cashier endorse Cheque

Define      FirstCopyOfBankDepositSlip And
            SecondCopyOfBankDepositSlip Is_A BankDepositSlip
```

```
End_Define

When        Cashier start Day
Then_Do     Cashier send Cheque to Bank And_Then
            Cashier file FirstCopyOfBankDepositSlip And_Then
            Cashier send SecondCopyOfBankDepositSlip to
            CreditDepartment

When        CreditClerk Of CreditDepartment receive
            SecondCopyOfBankDepositSlip from Cashier
Then_Do     CreditClerk write CustomerNumber Of Customer
            on SecondCopyOfBankDepositSlip And_Then
            CreditClerk send SecondCopyOfBankDepositSlip to
            DataPreparationSection

When        DataPreparationSection receive
            SecondCopyOfBankDepositSlip from CreditDepartment
Then_Do     DataPreparationSection store ReceiptData Of
            SecondCopyOfBankDepositSlip

When        AccountingDataProcessingDepartment end Day
Then_Do     AccountingDataProcessingDepartment process
            ReceiptData with AccountsReceivableMaster

When        AccountingDataProcessingDepartment end Month
Then_Do     AccountingDataProcessingDepartment print
            AccountsReceivableAgingSchedule And Statement
            And_Then AccountingDataProcessingDepartment send
            AccountsReceivableAgingSchedule to CreditManager
            Of CreditDepartment And_Then
            AccountingDataProcessingDepartment
            send Statement to Customer
```

<u>GSL specifications for Production Cycle</u>

Define ProductionPlanningAndControlDepartment
As PlanningDepartment
End_Define

When PlanningDepartment *receive*
Backorder **And** ProductionAuthorization
Then_Do PlanningDepartment *schedule* Backorder **And_Then**
PlanningDepartment *schedule* ProductionAuthorization

When ProductionSuperintended **And**
InventoryControlManagaer *compare*
ForecastedSalesOfProduct with
CurrentInventoryLevelOfProduct
If ForecastedSalesOfProduct <
CurrentInventoryLevelOfProduct
Then_Do ProductionSuperintended **And**
InventoryControlManagaer *prepare*
ProductAuthorization **And_Then**
ProductionSuperintended **And**
InventoryControlManagaer *send*
ProductionAuthorization to PlanningDepartment

When PlanningDepartment *determine*
QuantityRequiredMaterials **And** QuantityRequiredParts
After PlanningDepartment *receive*
Backorder **Or** ProductionAuthorization
If ((QuantityAtHand **Of** Materials > QuantityRequiredMaterials)
Or
(QuantityAtHand **Of** Materials = QuantityRequiredMaterials))
And
((QuantityAtHand **Of** Part > QuantityRequiredParts) **Or**

```
             (QuantityAtHand Of Part = QuantityRequiredParts ))
Then_Do     ProductionClerk schedule Backorder Or ProductAuthorization

When        PlanningDepartment determine
            QuantityRequiredMaterials And QuantityRequiredParts
            After PlanningDepartment receive
            Backorder Or ProductionAuthorization
If          (QuantityAtHand Of Materials <
            QuantityRequiredMaterials) And
            QuantityAtHand Of Part  < QuantityRequiredParts )
Then_Do     ProductionClerk prepare
            PurchaseRequisition And_Then
            ProductionClerk send PurchaseRequisition to
            PurchasingDepartment

When        ProductionClerk found ScheduleDateNear
Then_Do     ProductionClerk send Backorder Or ProductionAuthorization
            to DataPreparationSection Of
            ProductionDataProcessingDepartment

When        ProductionDataProcessingDepartment receive
            Backorder Or ProductAuthorization from PlanningDepartment
Then_Do     ProductionDataProcessingDepartment store
            ProductionAuthorization Or Backorder
            by ProductNumber And_Then
            ProductionDataProcessingDepartment prepare
            ProductionOrder And MaterialRequisition And
            MoveTicket And_Then
            (ProductionDataProcessingDepartment send
            MaterialRequisition to MaterialStoreroom) And
            (ProductionDataProcessingDepartment send
            ProductionOrder And MoveTicket to
            FirstProductionDepartment) And
```

(ProductionDataProcessingDepartment *send*
ProductionOrder to CostAccountingDepartment) **And**
(ProductionDataProcessingDepartment *store*
MaterialRequisition for InventoryProcessing)

Define FirstProductionDepartment **Is_A** ProductionDepartment
End_Define

Define FabricationDepartment **Is_A** FirstProductionDepartment
End_Define

When MaterialStoreroom *receive* MaterialRequisition from
ProductionDataProcessingDepartment
Then_Do MaterialStoreroom *send* Material to
ProductionDepartment **Of** MaterialRequisition

When ProductionDepartment *complete*
Backorder **Or** ProductionAuthorization
Then_Do ProductionDepartment *send* MoveTicket to
PlanningDepartment

When PlanningDepartment *end* Day
Then_Do PlanningDepartment *send* MoveTicket to
ProductionDataProcessingDepartment

When (ProductionDataProcessingDepartment *receive*
MoveTicket from PlanningDepartment) **And**
ProductionDataProcessingDepartment *end* Day
Then_Do ProductionDataProcessingDepartment *store*
MoveTicket by ProductionOrderNumber **And_Then**
ProductionDataProcessingDepartment *update*
OpenProductionOrderFile While
ProductionDataProcessingDepartment *prepare*

DailyProductionStatusReport

When MaterialStoreroom *end* Day
Then_Do MaterialStoreroom *send* MaterialRequisition to ProductionDataProcessingDepartment

When ProductionDataProcessingDepartment *receive* MaterialRequisition from MaterialStoreroom **And** ProductionDataProcessingDepartment *end* Day
Then_Do ProductionDataProcessingDepartment *store* MaterialRequisition by MaterialPartNumber

When WorkCenter *end* Day
Then_Do WorkCenter *send* JobTimeTicket to ProductionDataProcessingDepartment

When ProductionDataProcessingDepartment *receive* JobTimeTicket from WorkCenter
Then_Do ProductionDataProcessingDepartment *store* JobTimeTicket by ProductionOrderNumber

When ProductionDataProcessingDepartment *end* Day After (ProductionDataProcessingDepartment *store* MaterialRequisition by MaterialPartNumber) **And** (ProductionDataProcessingDepartment *store* JobTimeTicket by ProductionOrderNumber)
Then_Do ProductionDataProcessingDepartment *update* WorkInProcessMasterFile using MaterialRequisition **And** JobTimeTicket

When FabricationDepartment **Or** AssemblyDepartment *produce* Goods While FabricationDepartment **Or** AssemblyDepartment *process* Backorder **Or** ProductionAuthorization

Then_Do FabricationDepartment **Or** AssemblyDepartment
inspect Units **Of** Goods

When FabricationDepartment **Or** AssemblyDepartment
pass Units Of Goods While
FabricationDepartment **Or** AssemblyDepartment
inspect Units **Of** Goods
Then_Do FabricationDepartment **Or**
AssemblyDepartment *send*
Goods to MainWarehouse **And_Then**
FabricationDepartment **Or** AssemblyDepartment *send*
OrderRelease to PlanningDepartment **And**
CostAccountingDepartment

When PlanningDepartment *receive* OrderRelease from
FabricationDepartment **Or** AssemblyDepartment
Then_Do PlanningDepartment *record* Completion **Of** Backorder **Or**
ProductionAuthorization **And_Then**
PlanningDepartment *send* OrderRelease to
ProductionDataProcessingDepartment

When ProductionDataProcessingDepartment *receive*
OrderRelease from PlanningDepartment
Then_Do ProductionDataProcessingDepartment *store*
OrderRelease by ProductNumber **And_Then**
ProductionDataProcessingDepartment *update*
FinishGoodsMasterFile **And**
WorkInProcessMasterFile **And**
OpenProductionOrdersFile with OrderRelease **And_Then**
ProductionDataProcessingDepartment **prepare**
CompletedProductionOrdersReport

GSL specifications for Expenditure Cycle

Define Utilities **Is_A** Expenditure **End_Define**
Define Insurance **Is_A** Expenditure **End_Define**
Define Material **Is_A** PurchaseItem **End_Define**
Define Part **Is_A** PurchaseItem **End_Define**
Define Subassembly **Is_A** PurchaseItem **End_Define**

When ProductionClerk *compare* quantity **Of**
RawMaterialInventory with quantity **Of**
ProductionPlan for Item
If Quantity Of RawMaterialInventory for item <
Quantity Of ProductionPlan for Item
Then_Do ProductionClerk *create*
PurchaseRequisition **And_Then**
ProductionClerk *send* PurchaseRequisition to
PurchasingDepartment

WhenI nventoryControlClerk *check* RawMaterialInventory for item
If Quantity **Of** RawMaterialInventory for Item = ReorderLevel
Then_Do InventoryControlClerk **create** PurchaseRequisition **And_Then**
InventoryControlClerk *send* PurchaseRequisition to
PurchasingDepartment

Define InventoryControlClerk **As** InventoryControlClerk
End_Define

When Buyer **Of** PurchasingDepartment
receive PurchaseRequisition
Then_Do Buyer *select* Supplier who-is Reliable **And_Then**
Buyer *write* SupplierNumber And AcceptablePrice
for Item on PurchaseRequisition **And_Then**
Buyer *send* PurchaseRequisition to

DataPreparationSection **Of**
ProductionDataProcessingDepartment

When DataPreparationSection **Of**
ProductionDataProcessingDepartment *receive*
PurchaseRequisition from PurchasingDepartment
Then_Do ProductionDataProcessingDepartment *store*
PurchaseRequisition **And_Then**
ProductionDataProcessingDepartment *update*
RawMaterialInventoryFile **And** OpenPurchaseOrderFile
While ProductionDataProcessingDepartment prepare
PurchaseOrder by Supplier-number **And_Then**
Purchase-manager **Of** PurchasingDepartment *sign*
PurchaseOrder **And_Then** PurchasingManager
send PurchaseOrder to Supplier **And**
ReceivingDepartment **And** AccountsPayableDepartment
And_Then PurchasingDepartment *file*
PurchaseOrder by supplierName

When ReceivingDepartment *receive* PurchasingItem
Then_Do ReceivingClerk *retrieve* PurchaseOrder **And_Then**
(ReceivingClerk *count* Part) **Or**
(ReceivingClerk *weigh* Material) **And_Then**
ReceivingClerk *send* PurchasingItem to
MaterialStoreroom **And_Then**
ReceivingClerk *initial* receivingReport **And_Then**
ReceivingClerk *send* receivingReport to
AccountsPayableDepartment **And**
DataPreparationSection **Of**
ProductionDataProcessingDepartment **And_Then**
ReceivingClerk *file* ReceivingReport

When DataPreparationSection **Of**

```
           ProductionDataProcessingDepartment receive
           ReceivingReport from ReceivingDepartment
Then_Do    ProductionDataProcessingDepartment store
           ReceivingReport And_Then
           ProductionDataProcessingDepartment update
           RawMaterialInventoryFile by
           MaterialPartNumber And_Then
           ProductionDataProcessingDepartment update
           OpenPurchaseOrderFile by PurchaseOrderNumber

When       AccountsPayableClerk check SupplierInvoice with
           ReceivingReport And PurchaseOrder After
           AccountsPayableDepartment receive SupplierInvoice
Then_Do    AccountsPayableClerk prepare DisbursementVoucher
           And_Then AccountsPayableClerk record
           SupplierInvoiceData in
           VoucherRegister by PaymentDueDate

When       AccountsPayableClerk retrieve VoucherRegister
If         PaymentDueDate Of DisbursementVoucher = Today
Then_Do    AccountsPayableClerk send PaymentVoucher to
           DataPreparationSection Of
           AccountingDataProcessingDepartment

When       DataPreparationSection Of
           AccountingDataProcessingDepartment receive
           PaymentVoucher from AccountsPayableDepartment
Then_Do    AccountingDataProcessingDepartment store
           PaymentVoucher And_Then
           AccountingDataProcessingDepartment update
           AccountsPayableMasterFile While
           AccountingDataProcessingDepartment prepare
           ChequeVoucher And ChequeRegister by
```

```
            SupplierAccountNumber

Define      AccountingDataProcessingDepartment *update*
            AccountsPayableMasterFile **As**
            AccountingDataProcessingDepartment *credit*
            Amount Of PaymentVoucher to
            SupplierAccount **And_Then**
            AccountingDataProcessingDepartment *debit*
            Amount Of PaymentVoucher to
            AccountsPayableAccount
End_Define
```

Datacruncher Office Equipment, Inc.
Statement of Income
For the Years Ended December 31, 1991 and 1992

	(thousands of dollars)	
	1991	1992
Revenues		
Sales, dealers	$35,812	$30,654
Sales, direct	27,343	21,870
Service	7,327	6236
Total revenues	$70,482	$58,760
Cost of goods sold	49,934	39,375
Gross profit on sales	$20,548	$19,385
Operating expenses		
Selling and distribution expenses	$11,284	$ 9,532
Administrative (including accounting and data processing) expenses	2,302	1,875
Research and Engineering expenses	1,346	1,473
Interest expenses	1,372	1,013
Other expenses, including depreciation	741	733
Total operating expenses	$17,045	$14,626
Net income before income taxes	$ 3,503	$ 4,759
Provision for income taxes	1,191	1,618
Net Income	$ 2,312	$ 3,141

Figure A.1 Statement of income

Datacruncher Office Equipment, Inc. Statement of Financial Position December 31, 1991 and 1992		
	(thousands of dollars)	
	1991	1992
Assets		
Current assets		
Cash	$ 516	$ 2,178
Accounts receivable, net	12,022	9,518
Inventories		
Raw materials and parts	5,674	4,852
Work-in-process	6,923	5,107
Finished goods	9,547	7,321
Prepaid expenses	547	695
Total current assets	$35,229	$29,671
Fixed Assets	$12,184	$11,380
Less: Accumulated depreciation	7,132	5,939
Net fixed assets	$ 5,052	$ 5,441
Other assets	$10,636	$ 6,991
Total assets	$50,917	$42,103
Equities		
Current liabilities		
Notes payable	$ 5,731	$ 1,880
Current maturities of long-term debt	682	595
Accounts payable	4,619	3,751
Accrued expenses and taxes	6,978	4,826
Total current liabilities	$18,010	$11,052
Long term debt	$ 7,100	$ 6,400
Stockholders' equity		
Common stock, no par value		
Authorized 1,000,000 shares;		
outstanding 650,000 shares	$ 9,858	$ 9,858
Capital surplus	2,610	2,610
Retained earnings	13,339	12,183
Total stockholders' equity	$25,807	$24,651
Total equities	$50,917	$42,103

Figure A.2 Statement of financial position

Measures of Activity

DOCUMENT VOLUMES PER MONTH

Sales Orders (with an average of 6 items per document)	2000
Cash receipts	1950
Purchase requisitions	1000
Purchase orders (with an average of 8 lines per document)	1000
Back orders	260
Production orders	105
Shipping reports	1980
Bills of landing	1980
Materials requisition	475
Move tickets	355
Stock transfer notices	240
Receiving reports (with an average of 7 lines per document)	960
Disbursement vouchers	1010
Cheque vouchers (other than payroll)	830
NUMBER OF ACTIVE RECORDS IN KEY FILIES	
Accounts receivable	1850
Accounts payable	517
Finished-goods inventory	120
Work-in-process inventory	170
Raw-materials and parts inventory	11,960
Bills of material	120
Employee earnings	1098
General ledger	92
OTHER MEASURES OF ACTIVITY	
Number of new customers per month	20
Number of inquiries from customers per day	70
Number of adjustments (e.g. sales returns) per month	160
Number of days (on the average) between the time that purchase order is mailed and materials or parts are received	15
Number of days (on the average) required to process a sales order	12
Percentage of products rejected during production inspections during current year	5
Number of dealers accounting for 75% of sales by all dealers	40

Figure A.3 Measures of activity

APPENDIX (B)

AN EXAMPLE SYSTEM WITH EXECUTION

This appendix shows GSL specifications for a real world problem and the execution of specifications. This assignment case appears in appendix B in “Accounting Information systems: A Cycle Approach”[Davis 1990].

Bubbling Stone Beverage Company

Company Background

The Bubbling Stone Beverage Company began bottling its soft drink, Sassafras Ale. In 1895. The drink quickly began to enjoy strong local demand as a refreshing, tangy thirst quencher and gradually grew in popularity as a mixer for a variety of more libatious beverages. The company was then, and continues to be, a tightly controlled family business. The vast majority of the equity shares are held by Mr. Hamm R. Locke and the only son of the founder. Lesser number of shares are held by Mrs. Locke and three children. The company has always been led by a single strong personality (farther and son) who, fortunately, exemplified the personal attributes essential to successful leadership.

In the beginning, Bubbling Stone bottled its single product and distributed it through Associated Grocers, Inc. When Mr. Hamm R. Locke assumed control in 1950,the company entered a prolong period of sustained growth. Home-based in a large southern city, the company gradually expanded its operations to the 18 surrounding states. Mr. Locke first acquired the Popsi Cola franchies in his home city, followed in 1960 by the acquisition of the 6-UP and Dr. Peeper franchise. Company growth continued throughout the decades of the 1960s and 1970s through the acquisition of bottling and distributing operations in strategic locations in other states.

Today, all drinks are bottled exclusively in the home-based bottling facility and shipped to remote areas for distribution by other retailers and bottlers, and in an increasing number of instances by the company's own retail operations. This philosophy of centralized manufacturing and decentralized marketing has made the company one of the most cost-efficient bottling companies in the industry. The complete computerized high-speed production processes not only ensure highest standards of product quality, but it also permits distribution at retail prices significantly lower than competing colas and other brand-names soft drinks. Its line of products now includes Popsi, Diet Popsi, Popsi Free, Mountain Frost, 6-UP, Diet 6-UP, Dr. Peeper, Diet Dr. Peeper, A & Z Root Beer, Sunpict, Fruitrite, and its own patented Bubbling Stone Sassafras Ale. The company maintains separate records (by division) for its own local retail sales and sales to outside distributors.

Organization

As might be expected in a family company, all functions and operations fall under the personal supervision of the owner Mr. Locke. The organization is consisting of nine “departments” under the direct control of the President. The executive secretary to the president manages the flow of corporate information in and out of that office. Each department is headed by an executive of vice-president rank, to whom certain specific functions and responsibilities are delegated.

The vice-president of manufacturing and operations is responsible for procurement, manufacturing, and distribution of the product. Reporting to him are the director of operations, director of manufacturing, and the fleet manager.

The company vice-president manages the four retail divisions of the corporation. She has subordinates in charge of operations in Memphis, Atlanta, Mobile, and Jacksonville.

The vice-president/treasurer holds direct responsibility for all financial affairs of the company. His departmental operations are subdivided into general accounting, data processing, internal auditing, and office management.

The vice-president of marketing works directly with the public relation department and a special marketing unit to create and implement an overall marketing strategy for the company.

The vice-president of corporate development coordinates and test new products lines, whereas the vice-president of corporate affairs works within and outside the company to develop corporate goodwill and monitor the public image of the company.

Finally, the vice-president of sales holds direct responsibility of maintaining and monitoring the activities of the sales force.

Mr. Locke often humorously alludes to his “horizontal” organization, referring, of course, to the broad span of control at the executive level. He insists, however, that he is quite capable and comfortable with these divers and challenging responsibilities. In a speech to a local professional accounting meeting one evening, he stated that he personally “handled” every dollar that came into and went out of his business. He has set as his primary goal the acquisition of all major franchises for his company’s products in his home state and contiguous states.

System Logic (Revenue Cycle Subsystem)

In order to accomplish the corporation’s objectives, the president and his key officials have structured an effective network of transaction processing subsystems to handle, control, process, and report vital financial information. These subsystems are linked in an interdependent chain of information flows illustrated in Figure B.1. Although this chart portrays the company’s overall information systems philosophy, as described by the newly appointed director of internal auditing, specific operations may appear to fit only clumsily into the pattern. The following subsystems descriptions evolved from lengthy discussions with lower management and “line” personnel.

Sales

Bubbling Stone operates two separate sales systems: a “route” or city, sales system and an “outside” sales system in distinctly separate streams.

City Sales System

This system originates through the activities of route sales-persons, each traveling an established itinerary with a truckload of company products and calling on pre-designated established customers. At the beginning of each workday, the route sales-person’s truck is stocked with the products from inventory by the shipping department. He or she then given a list of approved customers on whom to call. Sales are made exclusively from the stock o the truck and may be made either for cash or on account. The sales person is authorized to pay for certain miscellaneous company expenses out of collected cash.

Check-in procedures at the end of the day require the salesperson to turn in all cash and unsold inventory and to account for all of the day’s transactions. A physical count of both return inventory and cash is made at once. The salesperson is required to reconcile total sales for the day with (1) cash turned in, (2) charge sales, and (3) out-of-pocket expenses. Discrepancies, if any, are identified and corrected before financial data are allowed to enter to the system.

When all route salespersons have “cleared” the check-in procedure, a master reconciliation is performed by a designated sales office clerk, and all sales related financial data enter to the *route sales system*. Data are processed to the general ledger system (Master database). Interfacing systems such as accounts receivable and inventory shipping are provided all information that is critical to their operation by the route sales system.

Outside Sales System

This system handles all sales to non-local bottlers and distributors. Orders are routinely taken by telephone, filled from finished goods inventory, and shipped via company fleet vehicle to remote locations.

Upon receipt of the telephone order, and before confirmation of the sale, inventories are verified and the customer's credit is checked. Confirmation is immediately followed by the creation of a four-part invoice by the person handling the sale. One copy is sent to accounts receivable. The remaining three copies go to shipping where one is filed and the other two travel with the product to the customer. The customer acknowledges receipt by signing and returning a copy of the invoice to accounts receivable via the truck driver. Accounts receivable then transmits sales data and customer account information to the general ledger system.

Accounts Receivable

The division of the sales function into route sales and outside sales subsystems has resulted in the maintenance of two separate accounts receivable subsystems. Customer account changes are independently transmitted to the general ledger system by each of the accounts receivable systems, and each acts as a subsidiary ledger supporting the general ledger control account. Payments received in the central mail room are divided according to type of account (route or outside) and are distributed intact to the appropriate accounts receivable systems. Cash handling is discussed under the section on cash control.

Shipping

The shipping function varies slightly in its service to route sales and outside sales.

Rout Sales

Each route salesperson provides a “stocking form” to shipping at the end of each day, specifying the goods he or she wants on the truck the following day. Each evening the shipping personnel withdraw the product from inventory and load the salesperson’s truck in accordance with the stocking form. The route sales system is then advised by an “executed” copy of the stocking form of the exact amount of inventory on the truck, which must be accounted for at the end of the following day.

Outside Sales

Shipping is notified of outside sales by the three-copy invoice set; a copy of the form serves as an inventory withdrawal document, and, as indicated earlier, two copies accompany the shipment to the customer. Shipping is responsible for obtaining a receipted (signed) copy of the invoice from the customer and returning it to accounts receivable-outside sales. This “trunaround” document then provides completed transaction evidence for inventory and general ledger system updating.

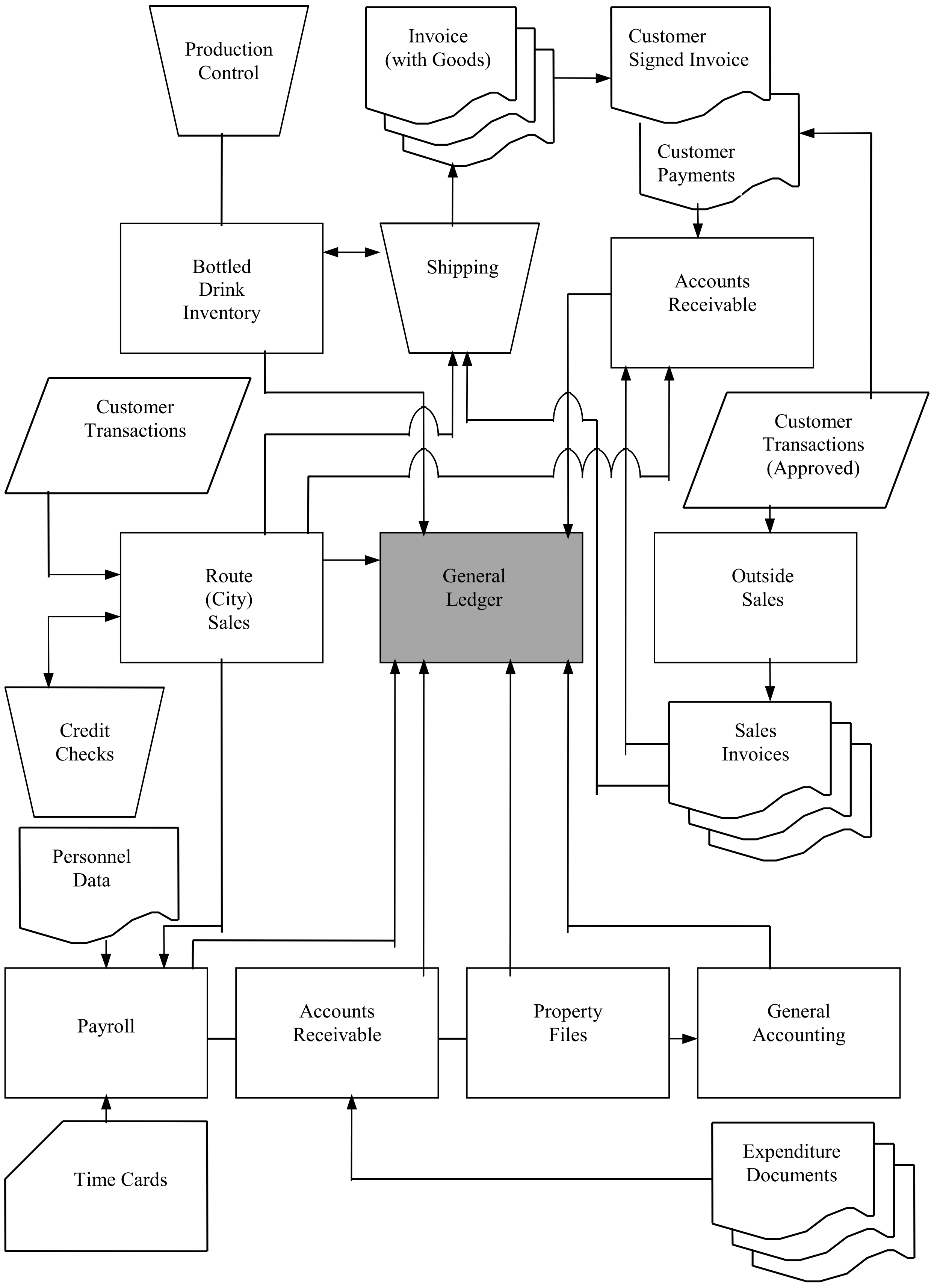


Figure B.1 Conceptual Framework of Financial Information System

Inventory

When bottled products are transferred to finished goods storage by production, the inventory and general ledger subsystems are notified and updated by production reports from production control. Although inventory withdrawals are made on the basis of (1) a stocking list for route sales and (2) copies of invoices for outside customer sales, it is incumbent on the two separate accounts receivable systems to provide transactional data to support the inventory subsystem and general ledger entries. The only direct link between inventory and the general ledger system occurs when reconciliations or manual adjustments are necessary.

In addition to the "Revenue Cycle Subsystem", Bubbling Stone company's financial system consists of three other subsystems namely, "Expenditure Cycle Subsystem", "Cash, Property, and General Ledger Subsystem" and "Production and Inventory Subsystem". However, the system logic for other three subsystems are not included in this section because only the "outside sales system" which is a part of the "Revenue Cycle Subsystem" is selected for illustration. Figure B.2 shows the overview of the outside sales system.

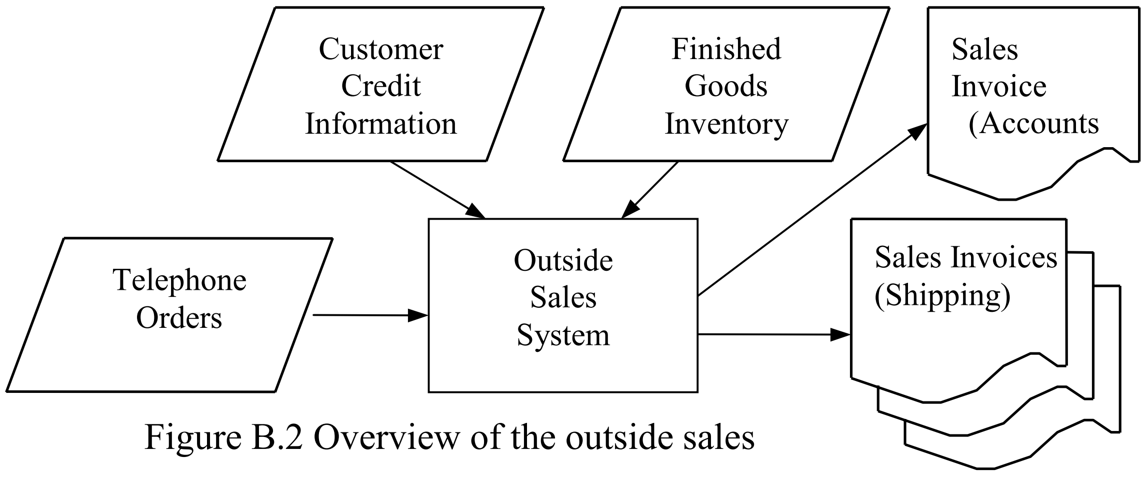


Figure B.2 Overview of the outside sales

Requirements

(1) Upon receipt of the telephone order, and before confirmation of the sale, inventories are verified and the customer's credit is checked.

(2) Confirmation is immediately followed by the creation of a four-part invoice by the person handling the sales. One copy is sent to accounts receivable. The remaining three copies go to shipping.

The above two requirements are extracted from the description given in the system logic of outside sales system. The activities at the shipping department upon receiving invoices are out of the boundary of the outside sales system. As we can see, these requirements just inform us what has to be performed by the outside sales system but do not say how it should be done. That is, we have incomplete information at this stage. Therefore the specification language should be capable to represent these incomplete requirements now, and later when the detail information are available it should also be capable of incorporating the additional information without major modifications to the existing specifications.

GSL specifications

In order to write GSL specifications, one has to identify the events, conditions and actions for each requirement. Table 4 depicts them for outside sales system. Although it is not clearly stated, the requirement (1) indicates that the outside sales system should reject the order in the case of invalid inventory or invalid customer's credit. If not, system should confirm the order.

Requirement	Event	Condition	Action
1	Receive Order		(1) Verify Inventory (2) Check Customer's Credit
1	Verify Inventory	Invalid Inventory	Reject Order
1	Check Customer's Credit	Invalid Credit	Reject Order
2	Order not rejected after inventory verification and customer's credit check		(1) Confirm Order (2) Create Invoice (3) Send One Copy of Invoice to Accounts Receivable (4) Send Other Three Copies of Invoice to Shipping Department

Table 4 Events, conditions and actions table for outside sales system

As shown in Table 4, there are three events for requirement 1 and one event for requirement 2. Following four GSL statements represent them.

(S1) **When** RecieveOrder
Then_Do InventoryVeirifcation **And** CustomersCreditCheck

(S2) **When** InventoryVerification
If InvalidInventory
Then_Do RejectOrder

(S3) **When**CustomersCreditCheck
If InvalidCredit
Then_Do RejectOrder

(S4) **WhenNot** RejectOrder **Just_After**
(InventoryVeirifcation **And** CustomersCreditCheck)
Then_Do ConfirmOrder **And_Then**
CreateInvoice **And_Then**
SendOneCopyToAccountsReceivable **And**
SendThreeCopiesToShipping

Specifications S1, S2 and S3 represent requirement 1 while specification S4 represents requirement 2. Note that, there is a user defined temporal relation called "Just_After". Suppose that the temporal relation "Just_After" is defined as follows:

(S5) If p **Just_Afte**r q
Then q **And_Then** p.

That is the reverse operation of **And_Then** connective. This simply improves the readability of the specification. The above specifications namely S1, S2, S3, S4 and S5 specify the most abstract level requirements of the outside sales system. The detailed information is left for later stage of the development. Therefore whatever the detailed specifications are, the above specifications remain unchanged. That means, we can specify detailed requirements without modifying the existing abstract level specifications. This proves the scalability of the specifications language. The abstract level specifications introduce the following abstract level objects:

RecieveOrder
InventoryVeirifcation
CustomersCreditCheck
InvalidInventory
RejectOrder
InvalidCredit
ConfirmOrder
CreateInvoice
SendOneCopyToAccountsReceivable and
SendThreeCopiesToShipping.

These are the most abstract level objects in the outside sales system. The view of a system adopted in this research allows creation of new objects using other objects and relations. Therefore the next step is to find out the next level objects and relations (major and minor relations) for the above objects. Table 5 provides a possible list of next level objects and relations.

Objects	Major Relations	Minor Relations
RecieveOrder OutsideSalesSystem Order	receive	
InventoryVeirifcation OutsideSalesSystem Inventory Order	verify	with
CustomersCreditCheck OutsideSalesSystem Customer CreditsDue Order	check Of (GSL relation)	in
InvalidInventory Inventory Product Quantity Order	greaterthan Of (GSL relation) Forall (GSL relation)	in
RejectOrder OutsideSalesSystem Order	reject	
InvalidCredit Order Customer CreditsDue Zero	greaterthan Of (GSL relation)	in
ConfirmOrder OutsideSalesSystem Order	confirm	
CreateInvoice OutsideSalesSystem Invoice	create	
SendOneCopyToAccountsReceivable SalesPerson Invoice One AccountsReceivable	send Of (GSL relation)	to copiesof
SendThreeCopiesToShipping SalesPerson Invoice Three Shipping	send Of (GSL relation)	to copiesof

Table 5 A possible list of objects and relations for outside sales system

Having the list of objects and relations in Table 5, the top level objects of the outside sales system can be defined as follows:

```
(S6)        Define      RecieveOrder As
                        OutsideSalesSystem receive Order
            End_Define

(S7)        Define      InventoryVeirifcation As
                        OutsideSalesSystem verify Order with Inventory
            End_Define

(S8)        Define      CustomersCreditCheck As
                        OutsideSalesSystem check
                        CreditsDue Of Customer in Order
            End_Define

(S9)        Define      InvalidInventory As
                        (Quantity Of Product in Order greaterthan
                        Quantity Of Product in Inventory)
                        Forall Product
            End_Define

(S10)       Define      RejectOrder As
                        OutsideSalesSystem reject Order
            End_Define

(S11)       Define      InvalidCredit As
                        CreditsDue Of Customer in Order
                        greaterthan Zero
            End_Define

(S12)       Define      ConfirmOrder As
                        OutsideSalesSystem confirm Order
            End_Define

(S13)       Define      CreateInvoice As
                        OutsideSalesSystem create Invoice
            End_Define

(S14)       Define      SendOneCopyToAccountsReceivable As
                        SalesPerson send One copiesof Invoice
                        to AccountsReceivable
            End_Define

(S15)       Define      SendThreeCopiesToShipping As
                        SalesPerson send Three copiesof Invoice
                        to Shipping
            End_Define
```

Specification S1 to S15 are the complete set of specifications for the outside sales system. It is clear from the above specifications that the changes to specifications S6 to S15 will not effect the specifications S1 to S5. That means the abstract level specifications remain unchanged even if the detailed specifications are changed. This guarantees the scalability of GSL specifications.

Transformation of GSL specifications to TANDTL logical statements

First, the top level objects in specifications S1 to S4 will be replaced by the bottom level objects and relations. The revised specifications will be:

```
(S1)  When      OutsideSalesSystem receive Order
      Then_Do   OutsideSalesSystem verify Order with Inventory And
                OutsideSalesSystem check
                CreditsDue Of Customer in Order
```

(S2) **When** OutsideSalesSystem *verify* Order with Inventory
If (Quantity **Of** Product in Order *greaterthan*
Quantity **Of** Product in Inventory) **Forall** Product
Then_Do OutsideSalesSystem *reject* Order

(S3) **When**OutsideSalesSystem *check*
CreditsDue **Of** Customer in Order
If CreditsDue **Of** Customer in Order
greaterthan Zero
Then_Do OutsideSalesSystem *reject* Order

(S4) **When**(OutsideSalesSystem *verify* Order with Inventory **And**
OutsideSalesSystem *check*
CreditsDue **Of** Customer in Order)
And_Then (**Not** OutsideSalesSystem *reject* Order)
Then_Do OutsideSalesSystem *confirm* Order **And_Then**
OutsideSalesSystem *create* Invoice **And_Then**
(SalesPerson *send* One copiesof Invoice
to AccountsReceivable **And**
SalesPerson *send* Three copiesof Invoice to Shipping)

Then the revised specifications S1 to S4 will be transformed to TANDTL logical statements.

(S1) receive(OutsideSalesSystem, Order) $\Rightarrow$
verify(OutsideSalesSystem, Order, Inventory) $\wedge$
check(OutsideSalesSystem, CreditsDue(Customer, Order))

(S2) verify(OutsideSalesSystem, Order, Inventory) $\wedge$
$\forall$Product. Greaterthan(Quantity(Product, Order),
Quantity(Product, Inventory))$\Rightarrow$ reject(OutsideSalesSystem, Order)

(S3) $\text{check(OutsideSalesSystem, CreditsDue(Customer, Order))} \wedge$
$\text{greaterthan(CreditsDue(Customer, Order), Zero)}$
$\Rightarrow \text{reject(OutsideSalesSystem, Order)}$

(S4) $[\text{verify(OutsideSalesSystem, Order, Inventory)} \wedge$
$\text{check(OutsideSalesSystem, CreditsDue(Customer, Order))}] \prod$
$[\neg\text{reject(OutsideSalesSystem, Order)}] \Rightarrow$
$\text{confirm(OutsideSalesSystem, Order)} \prod$
$\text{create(OutsideSalesSystem, Invoice)} \prod$
$[\text{send(SalesPerson, One, Invoice, AccountsReceivable)} \wedge$
$\text{send(SalesPerson, Three, Invoice , Shipping)}]$

For simplicity, let us rename objects as follows:

S	-	OutsideSalesSystem
O	-	Order
I	-	Inventory
D	-	CreditsDue
P	-	Product
Q	-	Quantity
C	-	Customer and
V	-	Invoice
SP	-	SalesPerson
SH	-	Shipping
AR	-	AccountsReceivable

Then the specifications S1 to S4 becomes

(S1) $\text{receive(S, O)} \Rightarrow \text{verify(S, O, I)} \wedge \text{check(S, D(C, O))}$

(S2) $\text{verify(S, O, I)} \wedge \forall P.\ \text{greaterthan(Q(P, O), Q(P, I))} \Rightarrow \text{reject(S, O)}$

(S3) $\text{check}(S, D(C, O)) \wedge \text{greaterthan}(D(C, O), \text{Zero}) \Rightarrow \text{reject}(S, O)$

(S4) $[\text{verify}(S, O, I) \wedge \text{check}(S, D(C, O))] \prod [\neg\text{reject}(S, O)] \Rightarrow$
$\text{confirm}(S, O) \prod$
$\text{create}(S, V) \prod [\text{send}(SP, \text{One}, V, AR) \wedge \text{send}(SP, \text{Three}, V , SH)]$

Inconsistency proofs

First, skolemization rules are applied to specification S2 to remove the variable P (Product). Then S2 will be :

(S2) $\forall P.\neg[\text{verify}(S, O, I) \wedge \text{Greaterthan}(Q(P, O), Q(P, I))] \vee \text{reject}(S, O)$

Because sub formulae verify(S, O, I) and reject(S, O) are free of variable P, we can move the universal quantifier to the beginning of the formula. Then we can simply ignore the universal quantifier. Therefore the following logical formulae can be used for non-clausal resolution.

(S1) $\neg\text{receive}(S, O) \vee [\text{verify}(S, O, I) \wedge \text{check}(S, D(C, O))]$
(S2) $\forall P.\neg[\text{verify}(S, O, I) \wedge \text{greaterthan}(Q(P, O), Q(P, I))] \vee \text{reject}(S, O)$
(S3) $\neg[\text{check}(S, D(C, O)) \wedge \text{greaterthan}(D(C, O), \text{Zero})] \vee \text{reject}(S, O)$
(S4) $\neg\{[\text{verify}(S, O, I) \wedge \text{check}(S, D(C, O))] \prod [\neg\text{reject}(S, O)]\} \vee$
$\{\text{confirm}(S, O) \prod \text{create}(S, V) \prod [\text{send}(SP, \text{One}, V, AR) \wedge$
$\text{send}(SP, \text{Three}, V , SH)]\}$

These statements can be rewritten as follows:

(S1) $\neg\text{receive}(S, O) \vee [\text{verify}(S, O, I) \wedge \text{check}(S, D(C, O))]$
(S2) $\forall P.\neg\text{verify}(S, O, I) \vee \neg\text{greaterthan}(Q(P, O), Q(P, I)) \vee \text{reject}(S, O)$
(S3) $\neg\text{check}(S, D(C, O)) \vee \neg\text{greaterthan}(D(C, O), \text{Zero})] \vee \text{reject}(S, O)$
(S4) $\neg\{[\text{verify}(S, O, I) \wedge \text{check}(S, D(C, O))] \prod [\neg\text{reject}(S, O)]\} \vee$

{confirm(S, O) ∏create(S, V) ∏ [send(SP, One, V, AR) ∧
send(SP, Three, V , SH)]}

Applying non-clausal resolution,

(R1) ∀P.¬receive(S, O)∨ ¬greaterthan(Q(P, O), Q(P, I)) ∨ reject(S, O)
⊥verify(S, O, I) (S1,S2)
(R2) ∀P.¬receive(S, O)∨ ¬greaterthan(Q(P, O), Q(P, I)) ∨ reject(S, O)
⊥check(S, D(C, O)) (S1,S3)
(R3) ¬receive(S, O) ∨ check(S, D(C, O)) ∨ {confirm(S, O) ∏create(S, V) ∏
[send(SP, One, V, AR) ∧ send(SP, Three, V , SH)]}
⊥verify(S, O, I) (S4,S1)
(R4) ∀P.¬verify(S, O, I) ∨ ¬greaterthan(Q(P, O), Q(P, I)) ∨
¬check(S, D(C, O)) ∨ ¬greaterthan(D(C, O), Zero)]
⊥reject(S, O) (S2,S3)
(R5) ∀P.¬greaterthan(Q(P, O), Q(P, I)) ∨ reject(S, O)∨ {confirm(S, O)
∏create(S, V) ∏ [send(SP, One, V, AR) ∧send(SP, Three, V , SH)]}
⊥verify(S, O, I) (S4,S2)

We can go on further, but will not be able to get "false" as the resolvent. That means we cannot prove that the set of logical formulae {S1, S2, S3, S4} is inconsistent.

Execution of GSL specifications

The executable specifications can be rewritten as follows:

(S1) Hold{receive(OutsideSalesSystem, Order)} ⇒
Exec{verify(OutsideSalesSystem, Order, Inventory) ∧
check(OutsideSalesSystem, CreditsDue(Customer, Order))}

(S2) Hold{verify(OutsideSalesSystem, Order, Inventory) ∧
∀Product. Greaterthan(Quantity(Product, Order),

Quantity(Product, Inventory))}$\Rightarrow$
Exec{reject(OutsideSalesSystem, Order)}

(S3) Hold{check(OutsideSalesSystem, CreditsDue(Customer, Order)) $\wedge$
greaterthan(CreditsDue(Customer, Order), Zero)} $\Rightarrow$
Exec{reject(OutsideSalesSystem, Order)}

(S4) Hold{[verify(OutsideSalesSystem, Order, Inventory) $\wedge$
check(OutsideSalesSystem, CreditsDue(Customer, Order))] $\prod$
[¬reject(OutsideSalesSystem, Order)]} $\Rightarrow$
confirm(OutsideSalesSystem, Order) $\prod$
Exec{create(OutsideSalesSystem, Invoice) $\prod$
[send(SalesPerson, One, Invoice, AccountsReceivable) $\wedge$
send(SalesPerson, Three, Invoice , Shipping)]}.

Suppose that the system has following databases:
Customer Database

Name	Address	Credits_Due
AA	Address AA	500
BB	Address BB	0

Inventory Database

Product	Quantity	Unit_Price
Popsi	4000	1

Order Database

Date	Order	Customer	Product	Quantity

Invoice Database

Date	Order	Customer	Product	Quantity

In addition to above databases, we need Action Database for execution.

Action Database

Action	Cycle

Now suppose that customer "AA" places an order number 1 for 300 Popsi on 98.04.01.

Execution steps

Cycle 0

System will initiate the functioning of the Outside Sales System by external event; customer AA's order and will create the record "receive(OutsideSalesSystem, 1)" in Action Database with the cycle number 0. At the cycle 0, the system will check the Action Database and find that there is an event "receive(OutsideSalesSystem, 1)" to be taken. Then the system will evaluate the action against the executable specifications. Only the condition in formula S1 is satisfied. Therefore system has to execute the actions "verify(OutsideSalesSystem, Order, Inventory)" and "check (OutsideSalesSystem, CreditsDue(Customer, Order))" in the next cycle. To do this, system will schedule these actions by creating records in Action Database as shown. The "1" in the cycle field indicates that the actions "verify(OutsideSalesSystem,1,Inventory)" and "check(OutsideSalesSystem, CreditDue(AA,1))" to be taken in the cycle 1. No more actions to be taken at cycle 0. Execution proceeds to cycle 1. Databases at the end of the cycle 0 will be as follows:

Customer Database

Name	Address	Credits_Due
AA	Address AA	500
BB	Address BB	0

Inventory Database

Product	Quantity	Unit_Price
Popsi	4000	1

Order Database

Date	Order	Customer	Product	Quantity
98.04.04	1	AA	Popsi	300

Invoice Database

Date	Order	Customer	Product	Quantity

Action Database

Action	Cycle
receive(OutsideSalesSystem, 1)	0
verify(OutsideSalesSystem,1, Inventory)	1
check (OutsideSalesSystem, CreditsDue(AA, 1))	1

Cycle 1

No new orders. The system will find that there are two actions "verify(OutsideSalesSystem,1,Inventory)"and "check (OutsideSalesSystem, CreditsDue(AA, 1))" to be executed in cycle 1. Then system will execute them.

While evaluating executable statements against the Action Database, system will find that condition of statement S2 is true as action “check (OutsideSalesSystem, CreditsDue(AA, 1))” is found in the Action Database and the CreditsDue of customer AA is greater than zero. Then the action “reject(OutsideSalesSystem, 1)” will be schedule for processing at the next cycle. Therefore system will generate a record in Action Database with the action “reject(OutsideSalesSystem, 1)” and “cycle 2”. Conditions of other statements are not true and no other actions to be taken at the cycle 1, processing will proceeds to cycle 2. The databases at the end of the cycle 1 are as follows:

Customer Database

Name	Address	Credits_Due
AA	Address AA	500
BB	Address BB	0

Inventory Database

Product	Quantity	Unit_Price
Popsi	4000	1

Order Database

Date	Order	Customer	Product	Quantity
98.04.04	1	AA	Popsi	300

Invoice Database

Date	Order	Customer	Product	Quantity

Action Database

Action	Cycle
receive(OutsideSalesSystem, 1)	0
verify(OutsideSalesSystem,1, Inventory)	1
check (OutsideSalesSystem, CreditsDue(AA, 1))	1
reject(OutsideSalesSystem, 1)	2

<u>Cycle 2</u>

Only one action to be performed. System will execute the action "reject(OutsideSalesSystem, 1)" and will find that none of the conditions in executable statements are true. Condition in statement S4 is not true because

[verify(OutsideSalesSystem, 1, Inventory) ∧
check(OutsideSalesSystem, CreditsDue(AA, 1))] ∏
[¬reject(OutsideSalesSystem, 1)]

is not held. At this cycle

[verify(OutsideSalesSystem, 1, Inventory) ∧
check(OutsideSalesSystem, CreditsDue(AA, 1))] ∏
[reject(OutsideSalesSystem, 1)]

is held because system "reject(OutsideSalesSystem, 1)" just after the actions "verify(OutsideSalesSystem, 1, Inventory)" and "check(OutsideSalesSystem, CreditsDue(AA, 1))". No more actions to be performed in this cycle. Processing moves to next cycle. Databases will remains unchanged.

<u>Cycle 3</u>

system will not find any actions to perform. No new orders for evaluation. Therefore system will stop processing.