The use of object oriented systems development methodologies in data warehouse development

Esterhuyse, J
12247219

Dissertation in partial fulfilment of the requirements for the degree Master of Science at the Potchefstroom campus of the North-West University.

Supervisor: Dr. R Goede

Co-supervisor: Prof. HM Huisman

Nov 2008

Abstract

CANDIDATE: Jacques Esterhuyse

PROMOTOR: Dr. R Goede / Prof. HM Huisman

DEPARTMENT: Computer Science

DEGREE: Master of Science (Information Technology)

KEYWORDS: Object oriented, data warehousing, decision support systems, data warehousing, information systems development methodologies.

Research has shown that data warehouses potentially offer great investment opportunities to business. To benefit from this, business needs to invest large sums of money. Such investments are very risky, as no guarantee of the success of these ventures can be given.

Object-oriented development has proved successful for developing operational systems in industry. This study researches object-oriented techniques to discover whether these techniques could be used successfully in data warehousing.

A literature study focuses on the definition of an information systems development methodology and defines the components of such methodology. A further literature study on four popular object-oriented methodologies determines the commonalities of these methodologies. In conclusion, a literature study on data warehouse methodologies is done to discover the phases and techniques used in developing data warehouses.

Based on the literature, a method is proposed to build a data warehouse harnessing object-oriented phases and techniques. The proposed method is applied as an interpretive experiment, followed by an evaluation of the data warehouse implemented.

Opsomming

KANDIDAAT: Jacques Esterhuyse

PROMOTOR: Dr. R Goede / Prof. HM Huisman

DEPARTEMENT: Rekenaar wetenskap

GRAAD: Meesters in wetenskap (Inligtingstegnologie)

SLEUTEL WOORDE: Objek georiënteerde ontwerp, data pakhuise, besluit steun stelsels, informasie stelsel ontwerp metodologie

Navorsing het getoon dat data pakhuise moontlik groot beleggings geleenthede beskikbaar stel aan besighede. Om die voordeel hiervan te realiseer moet besighede groot somme geld belê. Sulke beleggings is riskant aangesien die sukses hiervan nie gewaarborg kan word nie.

Objek georiënteerde ontwerp se sukses in die ontwikkeling van operasionele stelsels is reeds bewys in industrie. Met hierdie studie word navorsing gedoen na objek georiënteerde tegnieke om uit te vind of hierdie tegnieke suksesvol gebruik kan word in die ontwerp en skepping van data pakhuise.

'n Literatuurstude fokus op die definisie van 'n inligtingstelsel ontwikkelings metodologie en definieer die komponente van so 'n metodologie. 'n Verdere literatuurstudie is gedoen op vier populêre objek georiënteerde metodologië om die ooreenkomste te bepaal tussen die metodologië. Ten slotte is 'n literatuurstudie gedoen na data pakhuis metodologië om uit te vind watter fases en tegnieke gebruik kan word in die ontwikkeling van data pakhuise.

Gebasseer op die literatuurstudie, word 'n metode voorgestel vir die bou van 'n data pakhuis deur middel van objek georiënteerde fases en tegnieke. Die voorgestelde metode is toegepas as 'n interpratiewe eksperiment gevolg deur 'n evaluasie van die data pakhuis implementasie.

Table of contents

List of figures

List of tables

Chapter 1 - Introduction

1.1. Introduction

This chapter serves as an introduction to the study. It starts with a discussion on the underlying background and motivation for the study. Thereafter, the problem statement is established, followed by the methodology and limitations of the study. It concludes with the chapter allocation.

1.2. Background

"In today's world, the competitive edge is coming less from optimisation and more from the proactive use of information that these systems have been collecting over the years. Companies are beginning to realise the vast potential of the information that they hold in their organisations. If they can tap into this information, they can significantly improve the quality of their decision making and the profitability of the organisation through focused actions. The problem for most companies, though, is that their operational systems were never designed to support this kind of business activity, and probably never can be." (Anahory & Murray, 1997: 3). This statement proves that the information a company holds is potentially a great investment, but that money needs to be invested to reap the benefits thereof.

According to Anonymous (2006) in "Data Warehouse Portal Basics", a data warehouse (DW) is a tool to support the managing and the controlling business data. This is a system which is often at the heart of the strategic reporting systems. Its function is to consolidate and reconcile information from across disparate business units and IT systems and provide a means for reporting on, and analysing corporate performance management, profitability and consolidated financials compliance.

Anahory and Murray (1997:4) state that over the past 20 years, more than $1 trillion have been invested in new computer systems to gain competitive edge. This proves that developing an information system is a costly exercise, and one surely doesn't want it to be unsuccessful.

1.3. Motivation for study

Information systems methodologies are used to aid in information systems development. Boahene (1999) found that the use of methodologies enables the following when developing information systems:

- Provides background knowledge of the development – this includes the underlying assumptions, beliefs and values, as well as the nature of information systems
- Group development activities in process steps
- Provides transformation management - this is the needs of a client transformed into targeted information systems
- Techniques and methods to enforce standards and procedures used in the development of information systems

One such methodology is object-oriented (OO) development. Owing to its advantages, this methodology has grown very popular in software development. Avison and Fitzgerald (2003:247) point out that OO has the following advantages:

- It leads to a controlled environment due to concepts such as inheritance
- The organisation develops a library of object classes dealing with all the basic activities the organisation undertakes
- Classes get tested thoroughly in the component development phase, thus providing immediate industrial strength applications
- OO techniques are robust, error-free, quicker and cheaper

Ambler (2001:12) claims that the use of OO as a development methodology increases the chance of success for the following reasons:

- Provides models as a communication medium between users and the development team
- The use of models in turn allows one to work closely with the users of the system
- The time invested in defining the requirements and models pays off in the long run
- OO provides reusability for a wide variety of artefacts, such as code, models and components

Unfortunately, not all system environments are favourable for object-oriented approach. Ambler (2001:451) advises against object-oriented techniques for the following system environments:

- Systems for which structured techniques are ideal – It is argued that these systems are specifically built to fulfil only a certain role and no other
- Systems which cannot use OO throughout the entire development lifecycle – the reason for this being that the benefits of OO are achieved throughout the development lifecycle. Ambler warns that OO techniques should not be applied if the programming language does not support OO

From the above, it can be derived that OO is a very powerful methodology effectively managing risks in operational software development, though with certain limitations.

The second concept this study focuses on is data warehouses. Research has shown that data warehouse solutions are different to operational systems for the following reasons:

- Evolution and growth as business requirements for information change over a period of time (Anahory & Murray, 1997:8)
- Concepts of time variance and non-volatility essential for a data warehouse (Sen & Sinha, 2005:80)

Anahory and Murray (1997:8) point out that in practice, data warehouses must be designed to change constantly. They state that the main content of the data warehouse might be known, but it is unlikely to know all the detail required, the real problem being that the business itself may not be aware of its future information requirements. From the above, it can be derived that if a data warehouse solution is different from an operational solution, the development of such system should also be different from that of operational systems. This reflects in Anahory and Murray (1997:8) "in order to provide a flexible solution, one needs to look at the process that delivers a data warehouse, this process has to be fundamentally different from a traditional waterfall method"

Inmon (1996:260) advises against the use of the classical Systems Development Life Cycle (SDLC), also known as the waterfall approach.

From this, it can be assumed that the SDLC methodology will not produce the desired results in terms of developing a data warehouse, however there are methodologies other than the traditional information systems development methodologies (ISDMs) suitable for this purpose.

The above-mentioned methodologies are examined in Sen and Sinha (2005), comparing eight data warehouse methodologies. Sen and Sinha noted that as yet, none of the methodologies reviewed has achieved the status of a widely recognised standard, but two approaches well known in the development of data warehouses, are mentioned, i.e. the work of Inmon (1996) and Kimball *et al.* (1998).

Inmon's (1996:290) approach advocates the reverse of SDLC. Instead of starting from requirements, data warehouse development should be driven by data. Data is first gathered, integrated, and tested. Next, programs are written against the data and the results analysed. Finally, the requirements are formulated. The approach is iterative in nature. According to Anonymous (2006), the structure of the data warehouse is also different to Kimball's structure. Inmon follows a dependant mart structure, the top down approach. This approach transfers the data from diverse OLTP into a centralised area (the data warehouse). In this area, the data is organised into subject-oriented, integrated, non-volatile and time variant structures. The data marts are treated as subsets of the data warehouse and each data mart is built for an individual department. Once the data warehouse aggregation and summaries process is complete, it gets transferred to the staging area and a subset of transformations is done according to the departments' requirements. On completion of this process the OLAP environment gets refreshed.

The second approach is that of Kimball *et al.* (1998), the business dimensional lifecycle approach. This approach differs significantly from more traditional, data driven requirements analysis, and the focus is on analytic requirements elicited from business managers/executives to design dimensional data marts. The lifecycle approach starts with project planning, followed by business requirements definition, dimensional modelling, architecture design, physical design, deployment and other phases. Kimball's data warehouse structure follows what Anonymous (2006) calls the data warehouse bus structure, the bottom-up approach. The data marts are connected using a bus structure; this structure contains all the common elements used by data marts, such as conformed dimensions, measures, etc. defined for the enterprise and allowing one to query all data marts together. Kimball *et al.* (1998:19) defines the data warehouse as "nothing more than the union of all constituent data marts". The data flow in the Kimball's (1998) approach starts with extraction of data from operational databases into the staging area where it is processed and consolidated and thereafter loaded into the operational data store (ODS). Once the ODS is refreshed, the current data is once more extracted into the staging area and processed for the Data mart structure. Thereafter, the data from the Data mart is extracted to the staging area, aggregated, summarised and loaded into the Data Warehouse, from where it is made available to the end user for analysis.

From the above, it is clear that a data warehouse is a unique system, as it follows a unique development methodology and architecture. Object-oriented development, on the other hand, proved successful in industry, owing to characteristics such as reusability, polymorphism and inheritance. These characteristics provide great advantages, such as a better controlled environment, more robust solutions and a healthier financial outcome.

The study aims to investigate the applicability of object-oriented systems development methodologies and their techniques in the development of data warehousing systems.

1.4. Problem statement

The problem statement is to investigate the applicability of object-oriented information systems development methodologies and techniques in the development of data warehouses.

1.5. Methodology

Before focusing on the object-oriented data warehouse development objective, the key concepts of ISDMs need to be investigated and more specifically, the components of a methodology identified. Thereafter, an investigation into the object-oriented development approaches, as well as the approaches to a built data warehouse should be carried out. To compare methodologies and link data warehouse approaches to ISDMs, a proposed framework should be established. Iivari *et al.* (1999) proposes such a framework.

The study aims to understand data warehouse development methodologies and object-oriented development methodologies, in order to apply general object-oriented concepts in data warehouse development methodologies. Therefore the study will follow the action research approach illustrated in Figure 1-1.

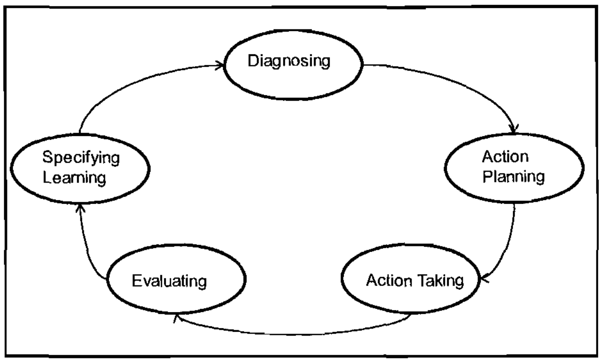


Figure 1-1 The Action Research Cycle (Baskerville, 1999:14)

This study will follow a qualitative research approach with an interpretive philosophy as the research methodology and secondly, follow action research as the research method

Information systems (IS) prototypes will be conducted in the form of an interpretive experiment by developing a data warehouse using an object-oriented approach. It will follow the development lifecycle of an object-oriented approach. The outcome of the case study should be to provide detailed information on components and concepts of OO successfully applied to the development of a data warehouse, as well as components and concepts not suited to this. This should also provide the possible advantages of developing a data warehouse in OO fashion, as well as the disadvantages of using such methodology.

1.6. Limitations of the study

The present-day data warehouse industry follows several approaches in data warehouse development. However, Sen and Sinha (2005) discovered that the approaches of Inmon (1996) and Kimball *et al.* (1998) are widely recognised in the development of data warehouses. Therefore, this study will concentrate on these data warehouse development approaches only. On the subject of object-oriented methods the study will focus on Object-Oriented Analysis (OOA), Object-Oriented Software Process (OOSP), Rational Unified Process (RUP) and Object Modelling Technique (OMT).

1.7. Provisional chapter allocation

Chapter 1 serves as an introduction to the study.

Chapter 2 provides a detailed discussion on the research methodology used in the context of information systems research.

Chapter 3 reports on a literature study covering information systems development methodologies. It focuses primarily on the components of systems development methodologies, such as philosophy, methodology, methods and techniques. Attention is given to a proposed classification system for information systems development methodologies.

Chapter 4 reports on literature relevant to object-oriented information systems development methodology. The focus of this chapter is on defining the components of object-oriented development, as well as reporting on most commonly used object-oriented methods.

Chapter 5 covers a literature study on data warehousing and reports on common development approaches used in data warehousing development.

Chapter 6 describes how to build data warehouses using object-oriented concepts and also serves as the research plan of the study.

Chapter 7 describes the implementation of the data warehouse as an interpretive experiment and serves as the action taking phase of the study.

Chapter 8 reports on the findings of the experiment and serves as the evaluation and specified learning of the study.

Chapter 2 - Research methodology

2.1. Introduction

The purpose of this chapter is to focus attention on the research methodology used in the study. It explains the commonly available research methodologies, as well as the reasons why the specific research methodology will be followed.

2.2. Overview of quantitative and qualitative research

Leedy and Ormond (2005:94) define quantitative research as an approach identifying relationships among measured variables with the purpose of explaining, predicting and controlling the phenomena.

In contrast, qualitative research is an approach that identifies the complex nature of a phenomenon with the purpose to describe or understand the phenomena from the participant's point of view.

Myers (1997:241) states that the reasoning behind preferring qualitative research over quantitative research is that qualitative methods are designed to help researchers understand people and the social and cultural contexts within which they live.

Kaplan and Maxwell (1994) also argue that the goal of understanding a phenomenon from the point of view of the participant and his/hers particular social and institutional context, is largely lost when textual data is quantified. Due to this reasoning, the study will follow a qualitative approach.

2.3. Philosophical perspectives

Myers (1997) argues that qualitative research consists of three subordinate philosophical epistemologies, illustrated in Figure 2-1.

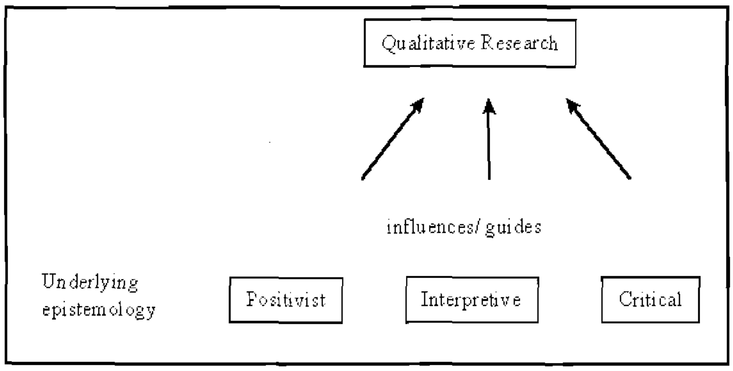


Figure 2-1 Underlying epistemology of qualitative research (Myers, 1997)

The underling philosophical epistemologies are:

- Positivist – This is stated to be when an attempt is made to test a theory. Orlikowski and Baroudi (1991:5) classify research as positivist if there was evidence of formal propositions, quantifiable measures of variables, hypothesis testing and the drawing of inferences about a phenomenon from the sample to a stated population.
- Interpretive – This is stated to be when an attempt is made to understand phenomena through the meaning people assign to them. Walsham (1993:4) defines interpretive research in information systems as "aimed at producing an understanding of the context on the information system, and the process whereby the information system influences and is influenced by the context". Klein and Myers (1999:72) suggest a set of principles for the conduct and evaluation of interpretive research, these being:
  - The fundamental principle of the hermeneutic circle.
  - The principle of contextualisation.
  - The principle of interaction between the researchers and the subjects.
  - The principle of abstraction and generalisation.

  - The principle of dialogical reasoning.
  - The principle of multiple interpretation.
  - The principle of suspicion.
- Critical – This is stated to be when an attempt is made to criticise a theory, in which the restrictive and alienating conditions of the status quo are brought to light. Harvey (1990:19) identifies the following shared elements in different critical methods: abstraction, totality, essence, praxis, ideology, structure, history, and deconstruction and reconstruction.

2.4. Qualitative research methods

Myers (1997) identifies four methods used when qualitative research is conducted. These are:

- Action research – This method is defined by Rapoport (1970:499) as a method that "aims to contribute both to the practical concerns of people in an immediate problematic situation and to the goals of social science by joint collaboration within a mutually acceptable ethical framework". As participative change is key to action research, it is often viewed as belonging to the critical social theory paradigm.
- Case study – This method describes a unit of analysis or a research method. A case study is an empirical inquiry that investigates a phenomenon within a real life context where the boundaries between phenomenon and context are not clear. Case study research methods are particularly well suited to IS research, as the focus is on information systems within the organisation.
- Ethnography – In this method, one is required to spend a significant amount of time in the field in order to study the phenomenon in its social and cultural context.
- Grounded theory – In this method, it is sought to develop theory that is grounded in data which is systematically gathered and analysed. Martin and Turner (1986)

define this as "an inductive, theory discovery methodology that allows the researcher to develop a theoretical account of the general features of a topic while simultaneously grounding the account in empirical observations or data." According to Myers (1997), the major difference between grounded theory and other methods is that its specific approach to the development suggests a continuous interplay between data collection and analysis.

Owing to the experimental nature of the study, it is recommended to use action research as a method for qualitative research.

2.5. Action research

Baskerville (1999:6) defines action research as a two step process:

- Diagnostic stage – defined as a collaborative analysis of the social situation by the researcher and the subjects of the research. During this stage, theories are formulated.
- Therapeutic stage – defined as a collaborative change experiment. The changes are introduced and the causes studied.

The following discussion is based on Baskerville (1999) and explains the approach to action research.

Most common action research approaches require a research environment and consist of a five phase cyclical process. Figure 2-2 shows the five respective processes.

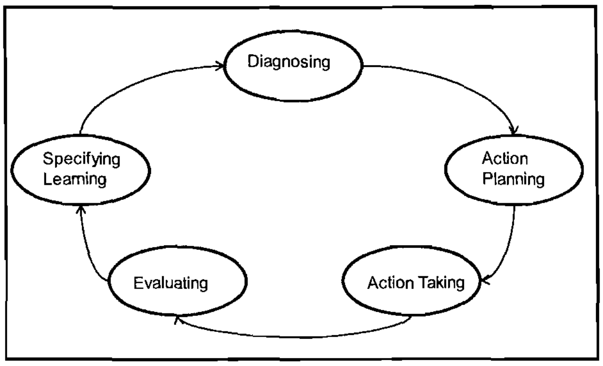


Figure 2-2 The Action Research Cycle (Baskerville, 1999:14)

The five processes are:

Diagnosing – is the process of identifying the primary reasons why change is necessary in the organisation. It involves self interpretation of the problem and should be done in a holistic fashion and not through reduction and simplification. The diagnosis should develop certain theoretical assumptions about the problem domain.

Action Planning – involves the researchers and practitioners to collaborate and produce actions that should relieve or improve the problems identified. A plan containing the necessary actions is created and executed by means of a theoretical framework. The plan should establish the target and approach for change.

Action Taking – implements the action plan. This causes changes in the organisation.

Evaluating – the outcomes of the plan implemented is evaluated. The evaluation determines whether the theoretical effects were realised and whether the problems identified are relieved or not. If the changes implemented were successful, it must be determined whether the changes were the sole cause of the success. If the changes implemented were unsuccessful, a framework for the next iteration should be established.

Specifying learning – is the knowledge gained from the research and may stem from the following sources:

- “Double-loop learning” – the knowledge gained from the restructuring of organisational norms to reflect new knowledge gained by the organisation during the research.
- If unsuccessful, the additional knowledge may provide a further foundation for diagnosis in preparation of further action research.
- The theoretical framework providing important knowledge for dealing with future research settings.

The action research cycle can continue irrespective of whether the action is successful or not.

Baskerville (1999:11) further explains that the ideal conditions for executing action research are:

- Settings where the researcher is actively involved, with the expectation of both the researcher and organisation benefiting
- Settings where knowledge obtained can be applied immediately
- Settings where research is a process of linking theory and practice

All the above conditions are applicable to the researchers study.

2.6. Research considerations for this study

The study aims to understand data warehouse development methodologies and object-oriented development methodologies, in order to apply common object-oriented concepts in data warehouse development methodologies.

Action research from a qualitative perspective will be used as research methodology.

As action research is applied, the study will follow the following research plan:

- Literature studies on systems development methodologies, object-oriented methodologies and data warehouse development methodologies
- Development and implementation of a data warehouse methodology incorporating object-oriented concepts and techniques
- Evaluation of the implemented data warehouse to determine whether the theory is a success

2.7. Summary

This chapter describes the type of research methodologies available. It starts with the difference between qualitative and quantitative research. The different types of philosophies found in qualitative research are positivist, interpretive and critical.

The qualitative research methods available are action research, case study, ethnography and grounded theory. Action research is discussed in more detail, as this is the preferred method for the researcher, while research considerations for this study are also covered.

Chapter 3 - Systems development methodologies

3.1. Introduction

This chapter introduces systems development methodologies. The main objective is to explain what is meant by a methodology and its components.

Methodologies do not have a universal definition, and this matter is regularly discussed by the information systems community (Avison & Fitzgerald, 2003:527).

Avison and Fitzgerald (2003:528) suggest that a methodology comprises of a number of components that specifies:

- How the project is broken down into stages.
- What tasks are to be carried out at each stage.
- What outputs are to be produced.
- When and under what circumstances, methodologies are to be carried out.
- What constraints are to be applied.
- Which people should be managed and controlled.
- What support tools may be utilised.

The authors also state that a methodology addresses the critical issue of a 'philosophy'. It is argued that this 'philosophy' gives methodology underlying theories and assumptions that shape the development of the methodologies. It gives a methodology unwritten aspects and beliefs that make the methodology effective in information systems development (ISD).

Huisman and Iivari (2003:1014) define a methodology as a combination of the following:

- A systems development approach(es):

This represents the philosophical view on which the methodology is built. It is the set of goals, guiding principles and beliefs, fundamental concepts and principles of the systems development process that drives interpretations and actions in systems development.

- A systems development process model(s):
  A process model as a representation of the sequences of stages through which a system evolves.
- A systems development method(s):
  A method is a systematic approach to conducting at least one complete phase of systems development, consisting of a set of guidelines, activities, techniques and tools, based on a particular philosophy of systems development and the target system.
- A systems development technique(s):
  Systems development techniques can be defined as a procedure, possibly with a prescribed notation, to perform a development activity.

Further study will be based on the definition of a methodology by Huisman and Iivari (2003). Components of the methodology will be dealt with in the following sections.

3.2. Systems development approach

According to (Iivari *et al.*, 2000:181), an information systems development approach (ISDA) is a class of methodologies which shares the fundamental concepts and principles for information systems development. On a more specific note, they define an ISDA as a set of related features that drives interpretation and action in information systems development (ISD). As previously discussed, an ISDA is a set of features: Goals, guiding principles and beliefs, fundamental concepts and principles for the ISD process. This can be described as follows:

- Goal – specifies the general purpose of the ISDA.

- Guiding principles and beliefs – form the common philosophy of the ISDA, which ensures that its information systems development methodology (ISDM) instances form coherent wholes.
- Fundamental concepts – largely define the nature of an information system (IS) implicit in the approach; the focus and unit of analysis in ISD.
- Principles for the ISD process – express the essential aspects of the ISD process in the ISDA.

Table 3-1 illustrates the feature of different ISDAs (Iivari *et al.*, 1999:4)

	Structured Approach	Information Modelling	Socio- technical Approach	Object-Oriented Approach	SSM Approach
Goals	To provide an approach that helps to produce high quality (reliable and maintainable) software in a productive way	To provide an approach for enterprise-wide development of information systems (databases) which enables co-ordinated development of integrated application systems and their long-term evolution	To provide an approach for ISD that enables future users to play a major part in the design of the system. To cater for job satisfaction objectives in addition to more technical and operational objectives, and to ensure that the new technical systems is surrounded by a compatible well-functioning organisational system.	To provide an approach which helps to ensure that the products are delivered to the user on time and within budget, that the products meet user requirements, that user requests to modify the system and/or fix bugs are responded to in a timely fashion, that increasingly sophisticated products are offered so as to keep a competitive edge, that the changes in standards and delivery technology are kept up and that the project team feels motivated and successful.	To provide a learning methodology to support debate on desirable and feasible changes.

Table 3-1 Summaries of the five IS development approaches (Iivari *et al.*, 1999:4)

	Structured Approach	Information Modelling	Socio- technical Approach	Object-Oriented Approach	SSM Approach
Guiding principles and b eliefs	Separation of the essential model from the implementation model; Careful documentation to make the development process visible; Graphical notations; Top-down partition able transformations / process models to hide complexity; Unambiguous, minimally redundant graphic specification; Balancing of models; Design modules with high cohesion and weak coupling	Data for a stable basis for information systems. Separation of conceptual and internal schemas. The conceptual schema forms the core model for an information system. Applications are built on top of the conceptual schema. IS development should be based on an engineering rigorous methodology.	Self-design of a work system; Minimal critical specification; Open-ended design process; Fit between the social and technical sub systems; Joint optimisation; Redundant functions.	Seamless analysis, design and implementation; Encapsulation; Information (implementation) hiding.	User of notional system modules called 'human activity systems' to illuminate different Weltanschauunge n which may be applied to any social system; An information system is a system to support the truly relevant human activity system.
Fund-amental con-cepts	Essential model vs. Implementation model; Transformation; Data flow; Data store; Terminator; Module; Cohesion; Coupling.	Universe of discourse. Information database; Conceptual schema; External schema; Entity Relationship; Attribute	Technical systems; Social systems; Variance; Unit operation; Technical needs; Social needs (job satisfaction)	Problem domain vs. Implementation domain; Object and class; Encapsulation; Information hiding; Inheritance; Polymorphism; Communication between objects	Weltanschauung; Human Activity Systems; Root definition; Relevant system.
Princi-ples for the ISD process	A step by step process at the detailed level of analysis and design activities. Situation dependent at the "strategic level" (waterfall or concurrent prototyping)	Incremental conceptual schema design; View integration.	User participation; Socio- technical design; Evolution.	Iterative and incremental development; Re-use	Stream of cultural analysis; Stream of logic-based analysis.

Table 3-2 (Continued) Summaries of the five IS development approaches (Iivari *et al.*, 1999:4)

To illustrate what is meant by the statement "an ISDA is a class of methodologies sharing fundamental concepts and principles", one can for instance examine the goals, guiding principles and beliefs, fundamental concepts and principles of object-oriented methodologies.

From the table above, the following features of an Object-Oriented Approach can be derived:

- The goal - To provide an approach which helps to ensure that products are delivered to the user on time and within budget, that the products meet user requirements, that user requests to modify the system and/or fix bugs are responded to in a timely fashion, that increasingly sophisticated products are offered so as to keep a competitive edge, that changes in standards and delivery technology are kept up and that the project team feels motivated and successful.
- Guiding principles – Seamless analysis, design and implementation; Encapsulation; Information (implementation) hiding.
- Fundamental concepts – Problem domain vs Implementation domain; Object and class; Encapsulation; Information hiding; Inheritance; Polymorphism; Communication between objects
- Principles for the ISD process – Iterative and incremental development; Re-use

Object-oriented analysis and design (OOAD) (Coad & Yourdon, 1991), Object Modelling Technique (OMT) (Rumbaugh *et al.*, 1991) and Rational Unified Process (RUP) (Jacobson *et al.*, 2001) essentially share the same attributes in terms of the goal, guiding principles, fundamental concepts and principles of the ISD process. These methodologies can be classified as methodologies based on the object-oriented approach.

Iivari *et al.* (1998:166) argues that ISDA may exist without any (methodology) instances and that it may serve as a template for deriving concrete ISDM instances.

3.3. Systems development process model

A software process model is a process of the sequence of stages through which a software product developed (Wynekoop & Russo, 1993:182). An example of this is illustrated in the system development life cycle (SDLC). The SDLC has many variants but follows the following basic structure and is executed in a sequential order (Avison & Fitzgerald, 2003:27):

- Feasibility study – This stage examines the present system and its intended requirements, the problems facing these requirements, new requirements and the investigation of alternatives.
- Systems investigation – This stage represents a fact finding mission. The following are examined: Functional requirements of the existing system and whether these requirements are being met. Requirements of the new system, constraints, exceptions and problems with the current working method. These facts are obtained through means of observation, interviews, questionnaires, searching of records and documentation and sampling.
- Systems analysis – Once the above facts are obtained, the analyst asks questions such as:
    - Why do these problems exist?
    - Why were certain methods of work adopted?
    - Are there alternative methods?
    - What are the likely growth rates of data?
- Systems design – This stage involves the design of both the computer and manual parts of the system.
- Implementation – In this phase the following aspects are addressed: quality control, education and training, documentation, such as operation and user manuals, and security. All of these need to be in place before implementation of the new system is allowed.

- *Review and maintenance – This is done while the system is operational.* The system is being evaluated for improvements on the current system.

Another example of a process model is the incremental approach (Avison & Fitzgerald, 2003:85). The first implementation is not seen as the main objective, but forms part of the continuing evolution and improvement of the original requirements.

Each iteration consists of a requirements-, analysis-, design- and implementation phase, all of which are repeated. The second iteration evolves iteration one and integrates it into the requirements of iteration two. The third iteration may reflect changes such as government imposed rules and mandatory changes. This iteration should result in a large portion of the requirements to be fulfilled. Figure 3-1 is a representation of the evolutionary development.

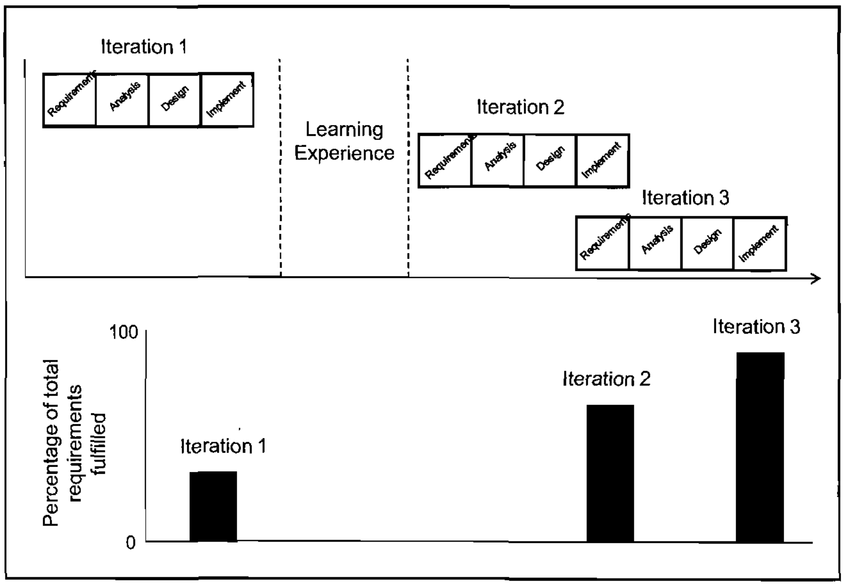


Figure 3-1 Evolutionary development (Avison & Fitzgerald, 2003:86)

According to Avison and Fitzgerald (2003:87) the spiral approach is a further attempt to combine the SDLC with the evolutionary process. Figure 3-2 illustrates the spiral process. The model adopts the concept of a series of incremental developments or releases. The development spirals outwards from the centre in a clockwise direction with each cycle of the spiral resulting in successive refinements of the system. The main activities in the spiral are planning (bottom left quadrant), determination of objectives (top left quadrant), risk analysis (top right quadrant) and development (bottom right quadrant). The model includes a risk phase to easily detect potential problems early in the process, before development is started.

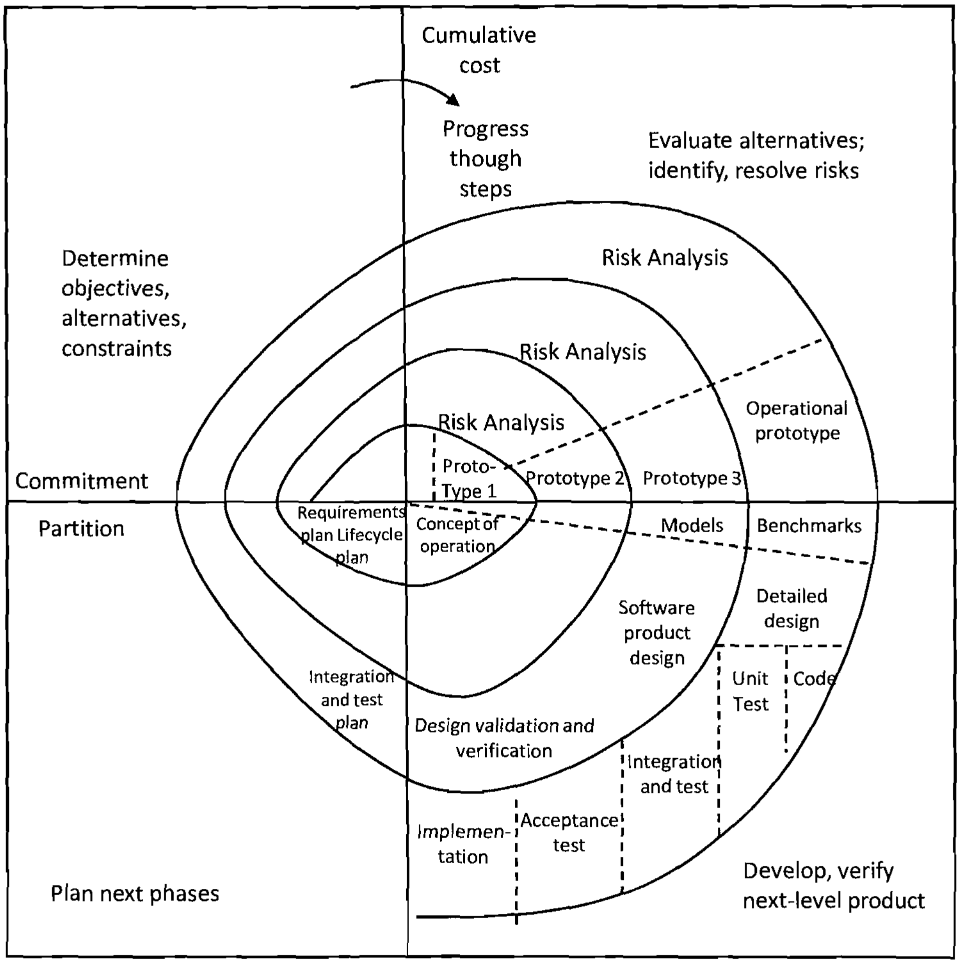


Figure 3-2 Boehm's spiral model (Avison & Fitzgerald, 2003:88)

3.4. Information systems development method

According to Wynekoop and Russo (1993:182) a method is a systematic approach to conducting at least one complete phase of systems development.

Typical examples of such phases are the design and testing of software, consisting of a set of guidelines, activities, techniques and tools of systems development pertaining to the target system.

From the above definition, one can derive that a method consists of a process model such as the SDLC, or an incremental or spiral process and, secondly, a set of tools and techniques. Examples of these are Object-Oriented Analysis and Design (method) and Soft Systems Methodology (method).

3.5. Systems development techniques

A systems development technique is a procedure with prescribed notation to perform a development activity. Brinkkemper *et al.*(1996:276) states that commonly, notations are referred to as techniques, but as in electrical engineering, there are standardised notations for transistors, resistors and the like. The application of these must follow a specific design of some structured plan. The same applies to software development. A technique relates to the type of development it supports.

As seen in Avison and Fitzgerald (2003:353), techniques are not necessarily unique in sets of methodologies, but can be shared across different methodologies, for example dataflow diagrams that are used in STRADIS and YSM. Another example is the use case technique that is used in Object-Oriented Analysis and Design (OOAD) and in Rational Unified Process (RUP) (Avison & Fitzgerald, 2003:413).

Iivari *et al.* (2000:180) argues that in order to cope with the confusion created by the proliferation of ISDMs, it is desirable to construct an organising structure that reduces the complexity of the myriad of ISDMs. This construct, the so-called dynamic classification framework, is discussed in a series of papers (Iivari *et al.*, 1998, 1999 and 2000).

3.6. Dynamic classification framework for classifying ISDM

The dynamic framework for classifying information systems development methodologies is illustrated in Figure 3-3 *(Iivari* et al.,2000).

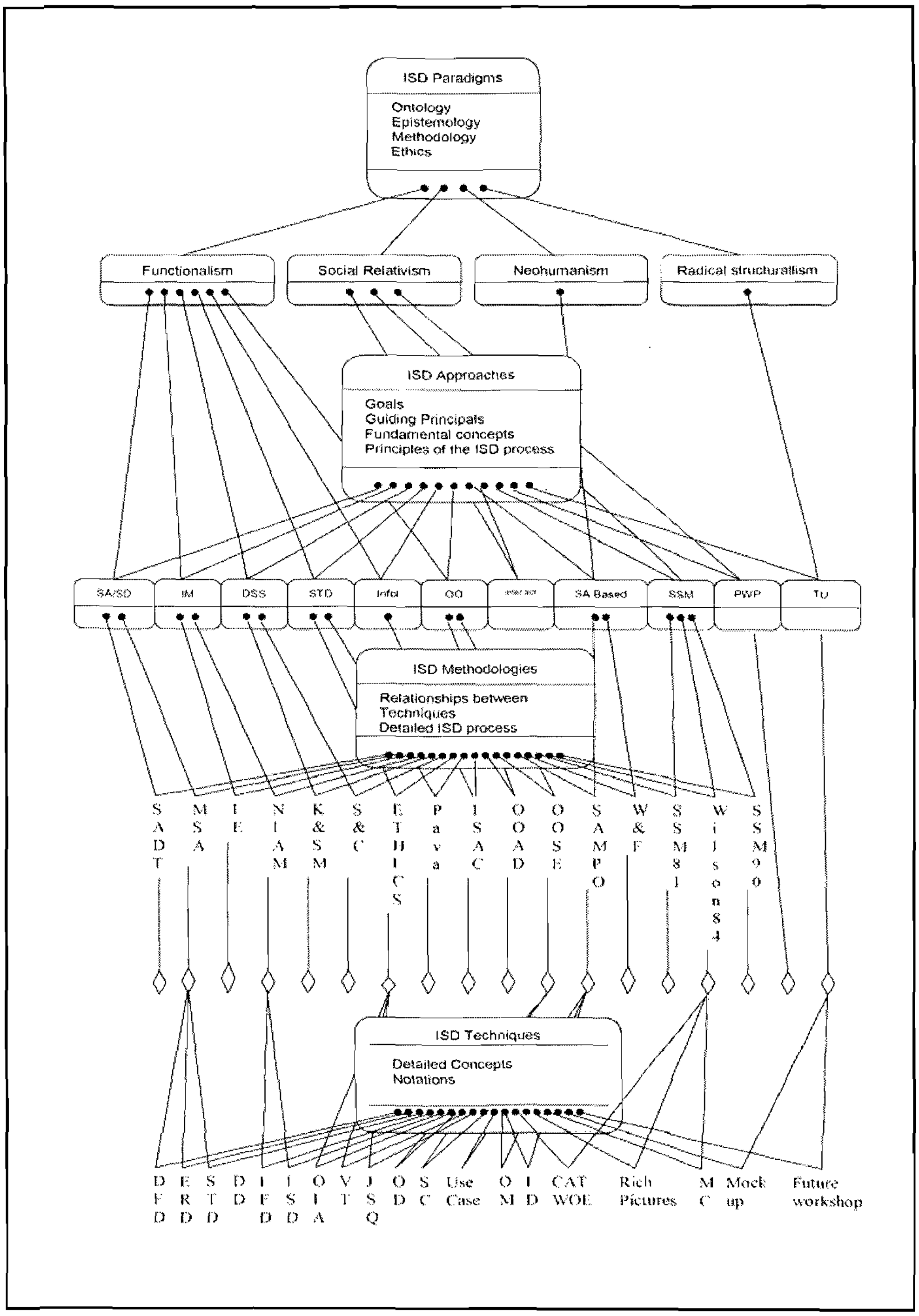


Figure 3-3 The dynamic classification framework (Iivari *et al.*, 2000:189)

This is a four tiered framework that concentrates on paradigmatic analysis rather than doing an analysis on the methodology level. It is stated that one should think of an ISDM as merely one instantiation of a more general abstract class, and that this class has the basic features that are inherited by all the ISDMs belonging to it. Iivari *et al.* (2000:181) argues that on the top level of the framework, one should find a set of philosophical (paradigmatic) assumptions and beliefs underlying every ISDA and ISDM. This makes it possible to group ISDM into paradigmatic positions.

It is also argued that there is a critical difference between Burrell and Morgan's (1979) and Kuhn's (1970) use of a paradigm. Kuhn uses it to describe the historical developments of natural science, whereas Burrell and Morgan use it to describe social sciences (Iivari *et al.*, 1998:171). The difference is that social sciences capture the basic assumptions of coexistent theories, whereas natural sciences capture the assumptions of historically successive theories. Information Systems (IS) research is more similar to social sciences than it is to natural sciences, and to differentiate Iivari *et al.* (1998:172) named it IS science. This indicates its paradigmatic status as an academic discipline rather like social sciences than natural sciences. It is also argued that IS science concentrates on three levels of analysis, namely individuals, organisations and society.

According to Iivari *et al.* (1998:172) the paradigmatic assumptions can be divided into four groups, namely:

- Ontology – This is assumed to be the nature of IS. It is proposed that the ontology of IS research is concerned with information and data, information systems, human beings in their different roles of IS development, the use of IS technology in the organisation, as well as the society. From this, one can infer what is to be the ontology assumption. Iivari *et al.* (1998:172) states that there are two types of ontology views, namely realism and idealism. Realism looks upon data as describing facts, information systems as consisting of technological structures, human beings as subject to casual laws and organisations as relatively stable structures. Idealism sees data as socially constructed meanings that signify intentions, information systems as a form of social systems realising human intentions, human beings as voluntaristic systems with consciousness and free will, technology as flexible structures subject to social and human choice and organisations as interaction systems or socially constructed systems.

- Epistemology – This is what human knowledge entails and how it can be acquired. There are two contrasting elements in epistemology according to Burrell and Morgan (1979), namely positivism and antipositivism. Positivism seeks to "explain and predict what happens in the social world by searching for regularities, casual relationships between its constituent elements". On the other hand, antipositivism "can only be understood from the point of view of the individuals who are directly involved in the activities which are to be studied. Antipositivism rejects the standpoint of the 'observer', which characterises positivist epistemology as a valid vantage point for understanding human activities. They maintain that one can only 'understand' by occupying the frame of reference of the participant in action. One has to understand from the inside rather than the outside" (Iivari *et al.*, 1999:5). Thus, according to Iivari *et al.* (1998:174), positivism views are scientific knowledge that consists of regularities, casual laws and explanations, whereas antipositivism emphasises human interpretation and understanding as constituents of scientific knowledge.
- Research methodology – Iivari *et al.* (1998:174) uses research methodologies in a context that refers to procedures used to acquire knowledge about ISDAs and related ISDMs methods and tools. The knowledge they refer to, is in the context of ISDAs consisting of rules and principles needed to elaborate and refine the ISDA. Iivari *et al.* (1998:175) divides research methods into three types, the first being constructive methods. This research method is concerned with the engineering of artefacts which may be purely conceptual artefacts, or more technical artefacts with a physical realisation. This method is highly emphasised in the IS- and computer science by March and Smith (1995), because it does not

describe existing reality, but rather create new ones. The second type is nomothetic methods. This includes format mathematical analysis, experimental methods and non experimental methods, such as surveys and field studies. The last type is idiographic methods where case studies and action research place "considerable stress upon getting close to one's subject and exploring its detailed background and life history (Burrell & Morgan, 1979:6)

- Ethics of research – This refers to the assumptions about the responsibility of the researcher for the consequences of his/her research approach and its results. Iivari *et al.* (1998:175) focuses on IS science as an applied discipline and IS as a practice. It distinguishes between two interrelated aspects; the role of IS as an academic discipline and the value of IS research. Three potential roles for IS science is identified: means-end oriented, interpretive and critical. In the first case, it is stated that the scientist aims at providing knowledge about means for achieving given goals, this without questioning the legitimacy of the goals. In the second case, the aim of an "interpretive scientist is to enrich peoples understanding of their action," "how social order is produced and reproduced" (Chua, 1986:615). Lastly, the critical scientist insists that research has "a critical imperative: the identification and removal of domination and ideological practice". Iivari *et al.* (1998:175) also states that when considering the value of IS research, one has to analyse whose and which values dominate the IS research, with the understanding that research may openly or latently serve the interests of particular groups. These groups could be top management, IS professionals, IS users and stakeholders.

The second level of the framework depicts the ISD approaches (ISDAs). Iivari *et al.* (2000:186) defines an ISDA as "a class of specific ISDMs that share a number of common features". The features of an ISDA are the following:

- Goals – specify the general purpose of the ISDA.

- Guiding principles and beliefs – form the common philosophy of the ISDA that ensures that its ISDM instances form coherent wholes.
- Fundamental concepts – define the nature of an IS implicit in the approach, this is the focus and unit of analysis in the ISD.
- Principles for the ISD process – express the essential aspects of the ISD process in the ISDA.

The third level of the framework represents the ISD methodologies (ISDM). From the framework two features are evident, one of which is the detailed ISD process. The definition of Iivari *et al.* (2000:186) fits this description. The definition defines an ISDM as “a codified set of goal-oriented procedures that guide the work and co-operation of the various parties (stakeholders) involved in the building of an IS application. These procedures are usually supported by a set of preferred techniques and tools, and guiding principles”.

The second feature is the relationship between techniques. Iivari *et al.* (2000:186) explains it as a sequence of elementary operations that more or less guarantee the achievement of certain outcomes if executed correctly.

ISDMs share the specific goals, guiding principles, fundamental concepts and principles of their respective approach (ISDA) (Iivari *et al.* 2000:188).

The last tier of the framework concentrates on the ISD techniques. As explained above, these are the techniques used in the ISDM. They are also seen as the lowest level of the framework.

The title of the study, “The use of object-oriented systems development methodologies in data warehouse development”, suggests that object-oriented development methodologies will be used. From the framework follows that one of the four products of the first tier (ISD paradigms) is functionalism and that one of the approaches in the functionalism paradigm is the object-oriented approach. Departing from here, the object-oriented approach and methodologies falling within it will be discussed in the next chapter.

3.7. Summary

This chapter introduced systems development methodologies. Information systems development methodologies were defined as a combination of systems development approaches (ISDA), systems development process models, systems development methods (ISDM) and systems development techniques.

The framework introduces the components of different methodologies. From the ISD paradigms tier, it is clear that the functionalism paradigm is one of the four paradigms illustrated. The object-oriented approach is part of this paradigm and the object-oriented methodologies part of the object-oriented approach.

In the following chapter, the object-oriented approach, as well as the methodologies falling within this approach, will be discussed. The chapter also compares the different object-oriented methodologies in an attempt to find a general use of object-oriented methodologies. This in turn will be used to map the object-oriented methodology to the different data warehouse development methodologies.

Chapter 4 - Object-Oriented Approach

4.1. Introduction

The purpose of this chapter is to focus attention on object-oriented (OO) approaches and methodologies. The chapter will start with a discussion on the OO approach.

The previous chapter introduced the features of systems development methodologies, as well as a framework for classifying methodologies. From the dynamic classification framework, it was determined that OO development is an approach to which OO methodologies belong. A literature study found that the Object-Oriented Analysis (OOA), Object-Oriented Software Process (OOSP), Rational Unified Process (RUP) and Object Modelling Technique (OMT) are popular OO methodologies. This chapter will therefore focus on these methodologies. The chapter also includes a comparison of these methodologies to find commonalities.

4.2. The object-oriented (OO) approach

The discussion is based on the lay-out of the dynamic classification framework (Iivari *et al.*, 2000) discussed in the previous chapter. The aspects of the methodology that will be discussed are the following:

- Definition and goal of the OO approach.
- The guiding principles and beliefs.
- Fundamental concepts.
- Principles of the ISD process.

4.2.1. Definition and goal of the OO approach

The goals of the OO approach are described by Iivari *et al.* (1999:4) as an approach which helps to ensure that

- products are delivered to the user on time and within budget.
- products meet user requirements.

- increasingly sophisticated products are offered to keep a competitive edge.
- the changes in standards and delivery technology are kept up.
- the project team feels motivated and successful.

4.2.2. Guiding principles and beliefs

The key points that Iivari *et al.* (1999:4) provides under guiding principles and beliefs are seamless analysis, design and implementation.

Avison and Fitzgerald (2003:247) explain that in OO one makes use of the unified modelling language (UML) to achieve this seamless analysis, design and implementation. UML is a set of rules and semantics that is used to specify the structure and logic of a system.

Booch *et al.* (2001:94) defines two categories for UML diagrams.

The first category is structural diagrams comprising the following:

- Class diagram – illustrates a set of classes, interfaces and collaborations and their relationships.
- Object diagram – illustrates a set of objects and their relationships. This serves to illustrate the data structures and static snapshots of instances of the objects found in class diagrams.
- Component diagram – illustrates a set of components and their relationships. This is used to illustrate the static implementation view of a system.
- Deployment diagram – illustrates a set of nodes and their relationships. It is used to illustrate the static deployment view of the architecture.

The second category of diagrams is dynamic behaviour diagrams. (Booch *et al.*, 2001:97). These are:

- Use case diagram – illustrates a set of use cases and actors and their relationships. This is used to illustrate the static view of a system. It is also used to organise and model behaviours of a system.
- Sequence diagram – an interaction diagram that emphasises the time ordering of messages. It shows a set of objects and the messages sent and received by these objects. This is used to illustrate the dynamic view of a system.
- Collaboration diagram – an interaction diagram that emphasises the structural organisation of the objects that send and receive messages. A collaboration diagram illustrates a set of objects, links among these objects and messages send and received by these objects.
- State chart diagram – illustrates a state machine, consisting of states, transitions, events and activities. State chart diagrams emphasise the event ordered behaviour of an object.
- Activity diagram – illustrates the flow from activity to activity within a system. An activity illustrates a set of activities, the sequential flow from activity to activity, as well as the object that acts and is acted upon. Activity diagrams are important for modelling the function of a system.

4.2.3. Fundamental Concepts

Iivari *et al.* (1999:4) identifies the following concepts as part of the OO approach:

- Problem domain vs. implementation domain – Ambler (2001:208) explains this as the problem space vs. solution space. Conceptual models are used to depict the detailed understanding of the problem space of the system. During the design, these conceptual models are evolved and furthered into classes that address the solution space and the problem space.
- Object and class – Booch (1994:83) defines an object as an entity that has a state, behaviour and identity. The structure and behaviour of similar objects are defined in their common class; the terms instance and object are interchangeable.

- Encapsulation information hiding – Booch (1994:50) defines encapsulation as the process of compartmentalising the elements of an abstraction that constitute its structure and behaviour; encapsulation serves to separate the contractual interface and an abstraction and its implementation.
- Inheritance – Ambler (2001:95) states that inheritance is a representation of an "is a", "is like" or "is kind of" relationship between classes. Avison and Fitzgerald (2003:243) argue that inheritance implies that the relationship is such that the hierarchy goes from classes of a general type down to classes of a more specific type.
- Polymorphism – Ambler (2001:173) explains that polymorphism enables objects to collaborate with other objects without knowing their type in advance.
- Communication between objects – (Ambler, 2001:161) Communication between objects is achieved by means of messaging; a message from one object to another object can be a request for information, or to execute a task.

4.2.4. Principles of the ISD Process

Iivari *et al.* (1999:4) states that the OO approach is iterative and incremental in its development process.

Ambler (2001:432) concurs with these principles by explaining that the waterfall approach does not truly reflect how software is developed and that a spiral approach is more realistic. The reason for this is that the spiral approach promotes iterative and incremental development. This allows development to be more suitable for changing business environments and getting portions of the development out quicker. The biggest disadvantage (Ambler, 2001:435) of iterative development is that it complicates the process of defining deadlines.

A second principle stated by Iivari *et al.* (1999:4) is re-use. This principle is found in concepts such as inheritance (Avison & Fitzgerald, 2003:247) and polymorphism. Once a class is created, it can be re-used time and again, thus avoiding 'reinventing the wheel' (Avison & Fitzgerald, 2003:146).

4.3. The applicability of the OO methodology

Ambler (2001:450) highlights the following system environments as an indication of when one should use OO development:

- Complex systems – it is argued that the easiest way to deal with complexity is to break it down into smaller components, and then deal with each component in turn. The OO paradigm is based on the concept of defining systems based on a collection of interacting objects. This strategy enables one to break down a complex system into smaller components.
- Systems that are prone to change – When the system is in its development stage, it is prone to change. It is argued that OO development leads to systems that are extensible.
- Systems with graphical user interface (GUI).
- Systems that are based on the client/server model.
- Systems that are integrated – With OO techniques one is able to develop wrappers around non-object technology. In OO style, this can be integrated with the organisation's overall system.

Not all system environments are favourable for OO approach Ambler (2001:451) advises against OO development for the following system environments:

- Systems for which structured techniques are ideal – It is argued that these systems are specifically built to fulfil a certain role.
- Systems which cannot use OO throughout the entire development lifecycle.

Avison Fitzgerald (2003:247) points out that OO has the following advantages:

- It leads to a controlled environment due to concepts such as inheritance.

- The organisation develops a library of object classes that deals with all the basic activities the organisation undertakes.
- Classes get tested thoroughly in the component development phase and therefore provide immediate industrial-strength applications.
- OO techniques are robust, error-free, quicker and cheaper.

4.4. The OO methodologies

There are numerous methodologies available, and it is not practical to discuss all of them. A literature study showed that the following OO methodologies are popular:

- Object-Oriented Analysis (OOA) (Coad & Yourdon, 1991).
- Object-Oriented Software Process (OOSP) (Ambler, 2001).
- Rational Unified Process (RUP) (Jacobson *et al.*, 1991).
- Object Modelling Techniques (OMT) (Rumbaugh *et al.*, 1991).

The following discussion will focus on the methodologies highlighted above.

4.4.1. Object-Oriented Analysis (OOA)

Coad and Yourdon (1991) are the original authors of the OOA methodology. The methodology was created before the unified modelling language (UML) existed and uses its own notation to describe objects and classes. This discussion will follow the original notation.

Coad and Yourdon (1991:178) define OOA as a method of analysis that identifies and defines the classes and objects found in the vocabulary of the problem domain.
The methodology consists of the following activities (Coad & Yourdon, 1991:34):

- Finding classes and objects.
- Identifying structures.
- Identifying subjects.

- Defining attributes.
- Defining services.

It is further explained that these steps should not be used as sequential steps, but regarded as the common overall approach. These activities, as discussed below, should be used iteratively.

4.4.1.1. Finding class and object

Booch (1994) expanded on the concepts of Coad and Yourdon (1991) for finding classes and objects. Booch (1994:155) explains three different approaches to find classes and objects, namely:

- classical approach.
- behaviour analysis.
- domain analysis.

The above approaches are discussed below:

Classical Approach

The classical approach is derived from principles of classical categorisation (Booch, 1994:155). Booch uses the work of Coad and Yourdon (1991) as an example of providing a source for potential objects. These are:

- Structure – the "Is a" and "part of" relationship.
- Other systems – an external system the application interacts with.
- Devices – devices the application interacts with.
- Event remembered – an historical event that must be recorded.
- Roles played – the different roles played in interacting with the system.
- Locations – the physical locations such as offices and sites important to the application.
- Organisational units – groups to which users belong.

Behaviour analysis

Behaviour analysis focuses on dynamic behaviour as the primary source of classes and objects (Booch, 1994:156). This is the knowledge the object maintains and the actions it performs.

The responsibilities of the object convey its purpose in the system. Classes are objects, grouped according to common responsibilities; this also forms hierarchies of classes.

Domain analysis

Domain analysis seeks to identify classes and objects common to all applications within the domain. Booch (1994:157) claims that domain analysis works well, except for unique kinds of software.

Moore and Bailin (1988:2) suggest the following steps for domain analysis:

- "Construct a straw man generic model of the domain by consulting with domain experts.
- Examine existing systems within the domain and present this understanding in a common format.
- Identify similarities and differences between the systems by consulting with domain experts.
- Refine the generic model to accommodate existing systems"

Booch (1994:158) explains that the domain expert can be the users of the system or their manager, but this will ultimately be the individual who uses the vocabulary of the problem domain.

4.4.1.2. Identifying structures

Coad and Yourdon (1991:79) define structure as an expression representing both the problem domain and the system's responsibilities. The term "structure" is used as an overall term to describe both generalisation-specialisation, or “gen-spec”- and “whole-part” structures.

The purpose of the structure is to focus on complexity of the problem and to uncover additional classes and objects that might not have been discovered (Coad & Yourdon, 1991:80). The gen-spec structure reflects a hierarchy of classes.

Coad and Yourdon (1991:84) suggest that, on the lower level classes, one should consider the following questions as a strategy for testing the gen-spec structure:

- Is it in the problem domain?
- Is it within the system responsibilities?
- Will there be inheritance?
- Will the specialisations meet the “what to consider and challenge” criteria?

The “whole-part” structure is hierarchies of objects indicating that one object is composed of, or made up from a series of sub-objects.

Coad and Yourdon (1991:90) suggest that one should consider three types of whole-part structures:

- The “assembly and it’s constitute parts”-type, i.e. an organisation and its departments.
- The “container and its contents”-type, i.e. a lecture hall and its seats.
- The “collection and its members”-type i.e. the football club and its players and helpers.

The set of criteria for testing the whole-part structure is similar to the test used for the gen-spec structure, with the exception that one does not test for inheritance in whole-part structures.

4.4.1.3. Identifying subject

Subjects are defined as a mechanism for guiding the reader (analyst, problem domain expert, manager and client) through a large, complex mode. Subjects are also helpful for organising work packages on larger projects, based upon initial OOA investigations (Coad & Yourdon, 1991:106).

As explained by Avison and Fitzgerald (2000:420), this is a bottom-up process with a top-down view. Grouping may be based on any criteria relevant to the area of concern; this can involve traditional, functional decomposition, but could also be based on problems or issues emerging from the problem domain.

One can use the example of a university problem domain. The subject layer can be admissions, courses, examinations and appeals. Admissions can be classes concerning applications, criteria, acceptance, references and payments.

4.4.1.4. Defining attributes

Coad and Yourdon (1991:119) define attributes as a form of data for which each object in a class has its own value. Coad and Yourdon (1991:121) suggest that the following steps be followed to identify attributes:

- Identify the attributes – Identify what the object in a class is responsible for knowing the value.
- Position attributes – Position which best describes the attribute within the class and object.
- Identify instance connections – This models the association between classes to manage complexity.
- Check for special cases – Special cases such as the following, need to be considered:
    - Check the attribute for a value of “not applicable”.

  - Check each class and attribute with just one attribute – should the class be considered or not?
  - Check each attribute for repeating values.
- Specify the attribute – Name the attribute according to the vocabulary used in the problem domain and the system's responsibility domain. Add descriptions for each attribute. Additional constraints will add to the description of the attribute.

4.4.1.5. Defining services

Coad and Yourdon (1991:143) explain that a service in an object is a specific behaviour that an object is responsible for exhibiting. The strategy used by Coad and Yourdon (1991:144), is to:

- Identify object states.
- Identify the requested services.
- Identify message connections.
- Specify the services.
- Put the OOA documentation set together.

OOA only focuses on the analysis phases of the solution and not on the design and implementation phases (Coad & Yourdon, 1991:178).

4.4.2. Object-Oriented Software Process (OOSP)

Ambler (2001:27) describes the OOSP methodology as a collection of process patterns. When brought together, these process patterns describe a complete process for developing, maintaining and supporting software.

OOSP uses the concept that large-scale, mission-critical, software development is serial in the large and iterative in the small. This leads to delivery of incremental releases of software on time.

Figure 4-1 is an overview of the interaction between techniques used in the OOSP methodology.

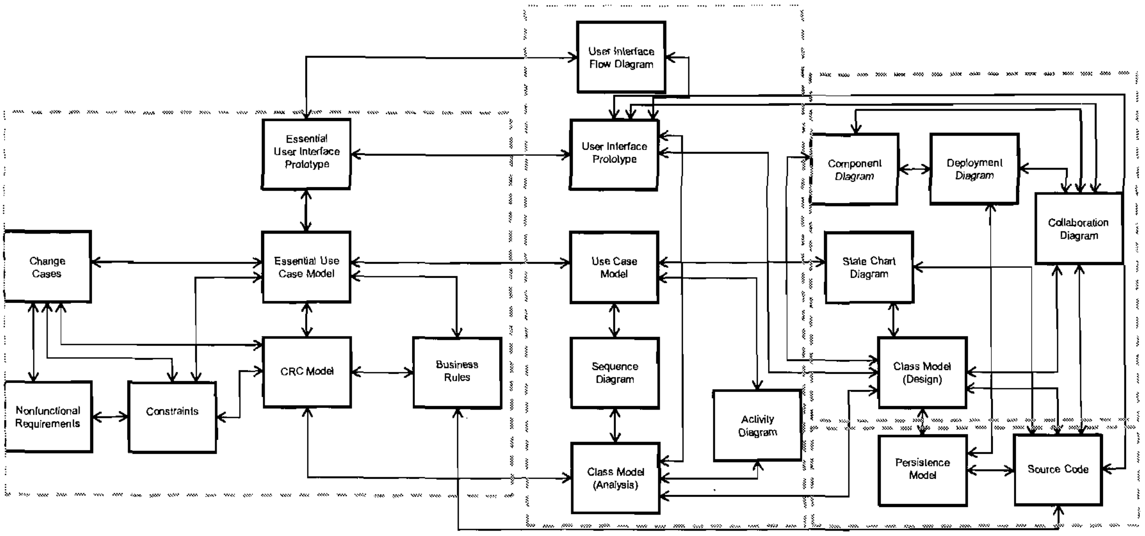


Figure 4-1 The OOSP Methodology (Ambler, 2001:439)

OOSP starts off by gathering the user requirements (Ambler, 2001:27) for the system and validating the requirements found (Ambler, 2001:110). This is shown in the left block in Figure 4-1. The requirements are analysed (Ambler, 2001:182), as illustrated by the middle block in Figure 4-1. The product of the analysis is then used for the design phase (Ambler, 2001:250), as indicated in the upper right block in Figure 4-1. Finally, the designs are implemented, as shown in the lower right block in Figure 4-1.

4.4.2.1. Gather requirements

The following discussion is based on Ambler (2001). The method starts with gathering the requirements for the system to be developed. A so-called requirement modelling team is put together. The team comprises subject matter experts (SME).

The SMEs are the individuals who are:

- Direct users of the system
- A customer / payer of the system

- Affected by the output of the system
- Required to approve the system
- Required to support the system

The purpose of the SMEs is to provide the analyst with the necessary requirements. Different techniques, such as interviewing or brainstorming, are used as gathering techniques.

The activities used for gathering requirements, are:

- Essential use case modelling
- Essential user interface prototyping
- Domain modelling

These activities are discussed in the following section.

Essential Use Case Modelling

Jacobson *et al.* (2001:122) distinguishes between two types of use case models:

- Firstly, an essential business, or abstract use case model. This is a technology-independent view of the behavioural requirements.
- Secondly, a system concrete, or detailed use case model. The function of this is to analyse the behavioural requirements describing in detail how users will interact with the system.

The essential use cases are firstly identified, and from these the system use cases are developed.

This discussion will focus only on essential use cases, as the function of these is to identify the essence of the problem in a technology-free, idealised and abstract environment (Ambler, 2001:52).

Figure 4-2 is an example of an essential use case model. The purpose of an essential use case model is to identify actions that provide measurable value to actors within a boundary and to depict the relationship between these entities.

The ellipse in Figure 4-2 represents a class. The link between the actor and the class is the relationship between the two entities and is described by a use case (Ambler, 2001:46).

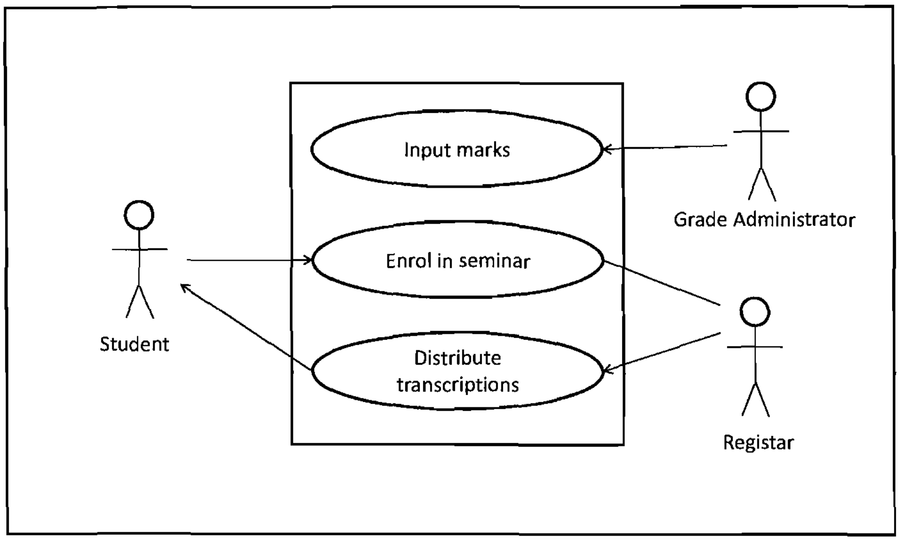


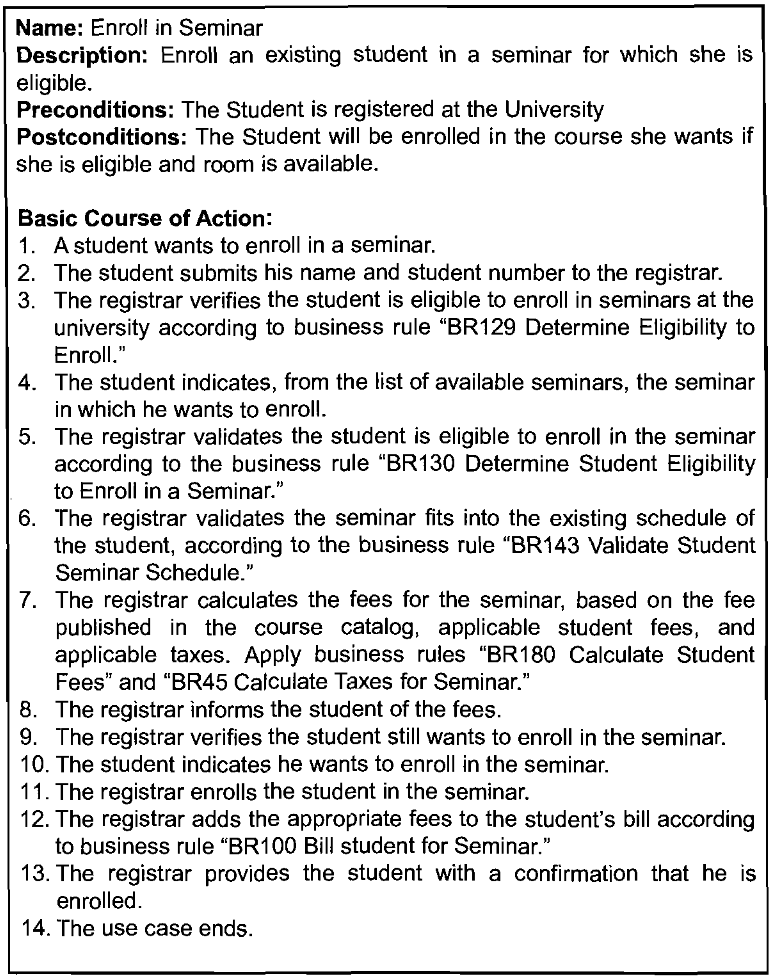
Figure 4-2 A use case diagram for a simple university (Ambler, 2001:46)

An actor represents anything that interacts with the system. To identify actors, one can follow the following questions:

- Who is the main customer of the system?
- Who obtains information from this system?
- Who provides information to the system?
- Who installs the system?
- Who operates the system?
- Who shuts down the system?
- What other systems interact with this system?
- Does anything happen automatically at a given time?

- Who will supply, use, or remove information from the system?
- Where does the system get information?

Once the essential diagram is completed, the essential use cases can be documented. A sample is illustrated in Figure 4-3.



Name: Enroll in Seminar
Description: Enroll an existing student in a seminar for which she is eligible.
Preconditions: The Student is registered at the University
Postconditions: The Student will be enrolled in the course she wants if she is eligible and room is available.

Basic Course of Action:

1. A student wants to enroll in a seminar.
2. The student submits his name and student number to the registrar.
3. The registrar verifies the student is eligible to enroll in seminars at the university according to business rule "BR129 Determine Eligibility to Enroll."
4. The student indicates, from the list of available seminars, the seminar in which he wants to enroll.
5. The registrar validates the student is eligible to enroll in the seminar according to the business rule "BR130 Determine Student Eligibility to Enroll in a Seminar."
6. The registrar validates the seminar fits into the existing schedule of the student, according to the business rule "BR143 Validate Student Seminar Schedule."
7. The registrar calculates the fees for the seminar, based on the fee published in the course catalog, applicable student fees, and applicable taxes. Apply business rules "BR180 Calculate Student Fees" and "BR45 Calculate Taxes for Seminar."
8. The registrar informs the student of the fees.
9. The registrar verifies the student still wants to enroll in the seminar.
10. The student indicates he wants to enroll in the seminar.
11. The registrar enrolls the student in the seminar.
12. The registrar adds the appropriate fees to the student's bill according to business rule "BR100 Bill student for Seminar."
13. The registrar provides the student with a confirmation that he is enrolled.
14. The use case ends.

Figure 4-3 "Enroll in seminar" as an essential use case (Ambler, 2001:55)

One way of identifying essential use cases, is to identify potential services by asking the SME the following questions from the actors' point of view:

- What are the users in this role trying to accomplish?
- To fulfil this role, what must users be able to do?

- What are the main tasks of users in this role?
- What information do users in this role need to examine, create or change?
- What do users in this role need to be informed of by the system?
- What do users in this role need to inform the system about?

Once all use cases are identified, they can be grouped into packages. This simplifies the complex diagrams.

Essential User Interface Prototyping

Essential user interface (UI) prototypes are UIs that are technology-independent, the purpose being to understand the requirements. The prototypes are done by using basic drawings.

Once all the UIs and the major components of the UIs are identified, a UI flow diagram is created. This diagram models the interactions between the users and the system for a certain use case, and it helps the analyst in getting a high level understanding of the UI for the system. Figure 4-4 is an example of a user interface flow diagram.

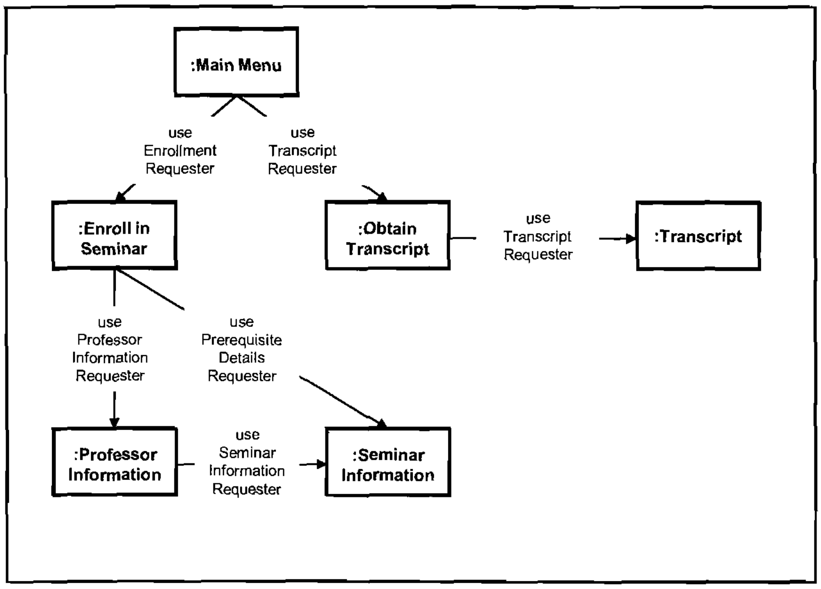


Figure 4-4 User interface flow diagram (Ambler, 2001:73)

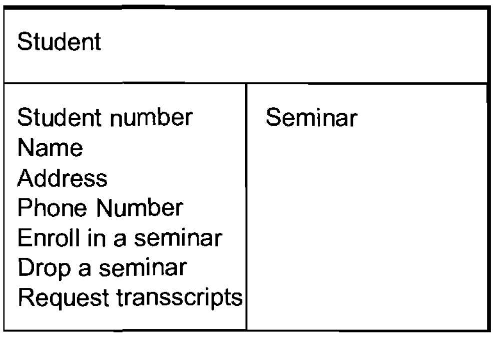
Domain Modelling

The function of domain modelling is to define the problem space. The problem space typically consists of classes representing the things and concepts within the domain in question.

Class Responsibility Collaborators (CRC) cards is a tool used to model classes. This tool is a collection of standard index cards divided into three sections:

- Class name – a collection of similar objects.
- Responsibilities – something that a class knows.
- Collaborators – what the other class needs to fulfil its responsibilities.

Figure 4-5 is a typical example of a CRC card. This example illustrates that the class is the Student, and its responsibilities are the student number, name, address, phone number, enrol in a seminar, drop a seminar and request transcripts. The collaborator for the student class is the seminar class.



Student	
Student number Name Address Phone Number Enroll in a seminar Drop a seminar Request transscripts	Seminar

Figure 4-5 An example CRC card (Ambler, 2001:76)

The domain modelling should be executed iteratively. The steps are:

- Find classes.
- Find responsibilities.
- Determine collaborators.
- Define use cases.

- Move the cards around according to the responsibilities.

There are three types of classes that exist, namely:

- Actor classes – representing the actors in the use case model and indicated by using "<<Actor>>" after the class name.
- Business classes – representing the places, things, concepts and events in the business.
- User interface classes – representing the screens and the menus in the system and indicated by using "<<UI>>" after the class name.

Develop a supplementary specification

The supplementary specification is a document containing all the requirements not specified in the use case model, user interface model, or the domain model. This document typically includes constraints, business rules and non-functional requirements.

Identify change cases

The change cases motivate the new requirements that have come forth, or the changes that need to be applied to the existing requirements. These changes should be documented.

Once the requirements are gathered, they should be validated to ensure that they are correct.

4.4.2.2. Validating the requirements

Ambler (2001:111) explains that the requirements can be misunderstood by the user, the analyst, or the designer. This is why requirements should be validated, thus ensuring accuracy.

It is recommended that testing should be done early and often to avoid problem fixing at a later stage, as this can be a costly exercise (Ambler, 2001:111).

The following discussion is based on Ambler (2001). The requirements can be tested by using the following techniques:

- Use case scenario testing.
- User interface walkthroughs.
- Requirements reviews.

Use case scenario testing

This is a technique that tests the domain model, the CRC model, or a class model. The process is as follows:

- Perform domain modelling – create a CRC model, or an analysis class mode that represents the domain.
- Create the use case scenario – it describes the situation which the system may, or may not be able to handle. The use case scenario is different to the use cases in the sense that it describes the logic, including the basic and alternative course of action.
- Assign classes to SMEs – one or more classes should be assigned to each SME in order to spread the functionality of the system to every SME.
- Act out a scenario – the SMEs act on the rolls of the cards given to them. This serves to describe the business logic of the responsibilities of each use case. A ball is used to indicate that the class or SME is busy processing; the ball is passed on to the next class or SME, it is collaborating with.
- Update the domain model – during testing, the missing responsibilities will be highlighted, thereby allowing updating of the domain model.

User interface walkthroughs

The user interface walkthroughs are similar to the user case walkthroughs, the only difference being the UIs are tested and not the domain models. The SMEs describe which screen and component on this screen will be used when a scenario is acted out.

Requirements review

The requirements review is a process in which a facilitator reviews the requirements gathered by the stakeholders responsible for the system. This is to verify that the requirements gathered, are correct and that the needs of the users are fulfilled.

Once the requirements are verified, the analysis of the requirements can be executed.

4.4.2.3. Object-oriented analysis (OOA)

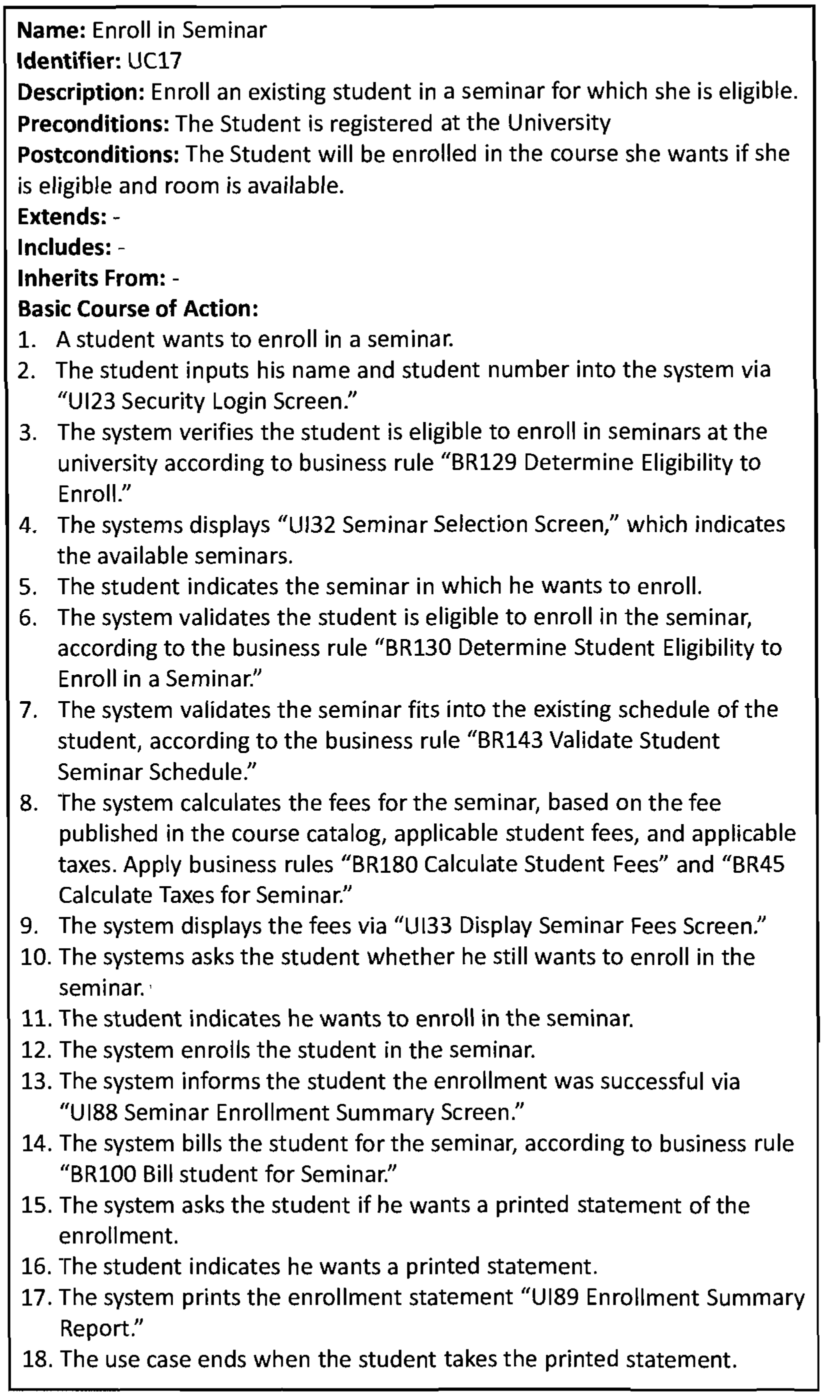
One should to keep in mind that the OOA used in OOSP, is used in the context of a phase. This phase is based on the OO methodology originally developed by Coad and Yourdon (1991) as discussed in section 4.4.1

Ambler (2001:182) explains that the purpose of analysis is to understand what needs to be developed. Analysis is an iterative process that is highly interrelated to requirements gathering. The essential models created in the requirements gathering are evolved into their corresponding analysis artefacts. The following is based on Ambler (2001:185).

System use case modelling

The essential use case model is evolved into a system use case. It is similar to the essential use case with the exception that it includes high-level implementation decisions, such as the screen numbers and properties, for example “extends”, inherited.

Figure 4-6 is an example of the essential use case “Enroll in seminar” evolved into a system use case.



Name: Enroll in Seminar
Identifier: UC17
Description: Enroll an existing student in a seminar for which she is eligible.
Preconditions: The Student is registered at the University
Postconditions: The Student will be enrolled in the course she wants if she is eligible and room is available.
Extends: -
Includes: -
Inherits From: -
Basic Course of Action:

1. A student wants to enroll in a seminar.
2. The student inputs his name and student number into the system via "UI23 Security Login Screen."
3. The system verifies the student is eligible to enroll in seminars at the university according to business rule "BR129 Determine Eligibility to Enroll."
4. The systems displays "UI32 Seminar Selection Screen," which indicates the available seminars.
5. The student indicates the seminar in which he wants to enroll.
6. The system validates the student is eligible to enroll in the seminar, according to the business rule "BR130 Determine Student Eligibility to Enroll in a Seminar."
7. The system validates the seminar fits into the existing schedule of the student, according to the business rule "BR143 Validate Student Seminar Schedule."
8. The system calculates the fees for the seminar, based on the fee published in the course catalog, applicable student fees, and applicable taxes. Apply business rules "BR180 Calculate Student Fees" and "BR45 Calculate Taxes for Seminar."
9. The system displays the fees via "UI33 Display Seminar Fees Screen."
10. The systems asks the student whether he still wants to enroll in the seminar.
11. The student indicates he wants to enroll in the seminar.
12. The system enrolls the student in the seminar.
13. The system informs the student the enrollment was successful via "UI88 Seminar Enrollment Summary Screen."
14. The system bills the student for the seminar, according to business rule "BR100 Bill student for Seminar."
15. The system asks the student if he wants a printed statement of the enrollment.
16. The student indicates he wants a printed statement.
17. The system prints the enrollment statement "UI89 Enrollment Summary Report."
18. The use case ends when the student takes the printed statement.

Figure 4-6 System use case (Ambler, 2001:187)

Sequence diagram

Sequence diagrams are developed from the use cases. Jacobson *et al.* (2001:251) states that the function of sequence diagrams is to model the logic of usage scenarios. A usage scenario is a description of a potential way in which the system can be used. This may include use cases or alternative courses and provides a bridge between the use cases and the class models.

Figure 4-7 shows a typical sequence diagram. The boxes at the top represent classifiers or instances, which can be use cases, objects, classes or actors. The lines from the top boxes represent object lifelines, meaning the life span of the object during the scenario being modelled. The long thin boxes on the lifelines are method-invocation boxes. They indicate that a process is being performed on the given object to fulfil a message. Messages are represented by labelled arrows.

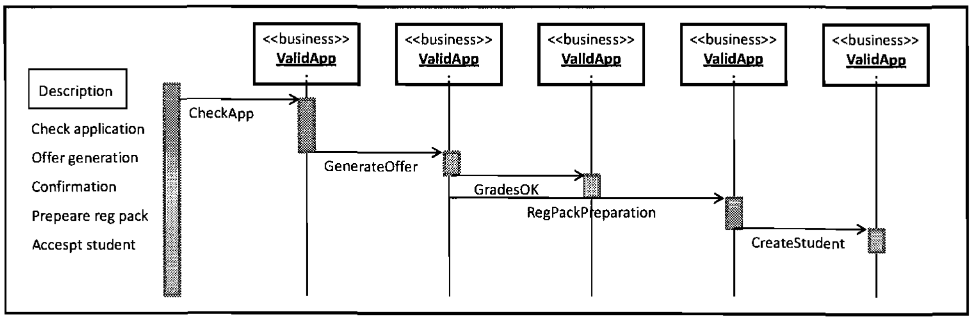


Figure 4-7 Sequence diagram for student (Avison & Fitzgerald, 2000:199)

Conceptual modelling

Class diagrams represent the conceptual model. The function of class diagrams is to model the classes of the system, the relationships between them, as well as their operations and attributes. The conceptual model is used to depict a detailed understanding of the problem space. During the design phase, the model is evolved to include classes that address the solution space.

The class model contains the following elements:

- Classes

- Methods
- Attributes
- Associations
- Dependencies
- Inheritance relationships
- Aggregation associations
- Association classes

Figure 4-8 illustrates a typical UML class diagram.

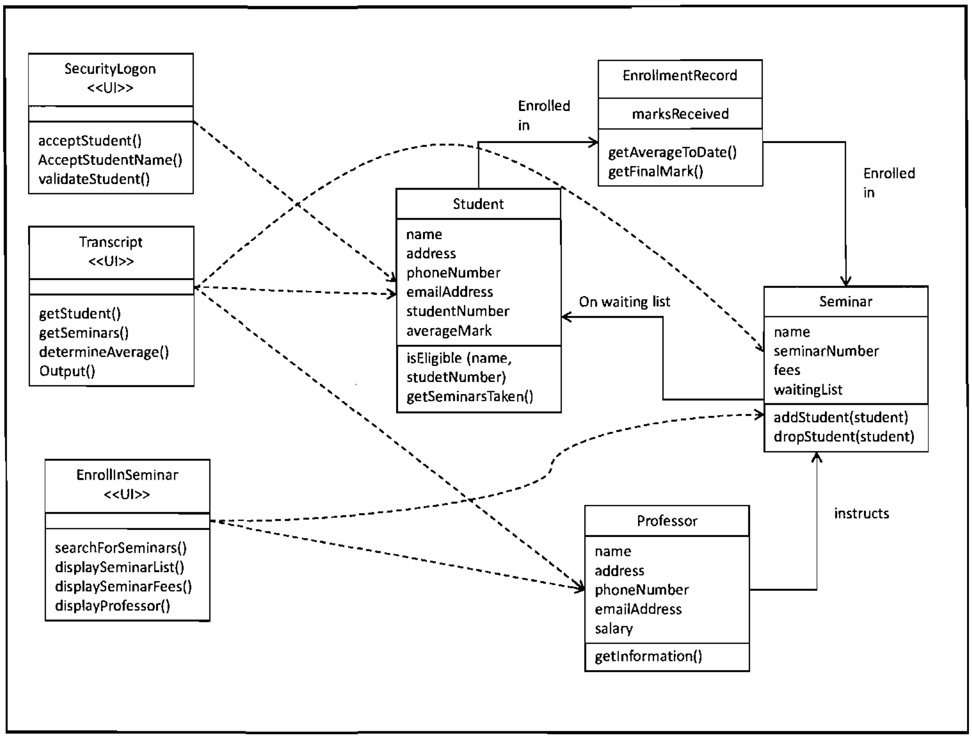


Figure 4-8 A UML class diagram based on the CRC model (Ambler,2001:210)

Each rectangle represents a class, with its name on top. Below the class name is the stereo type indicated by “<<” and ”>>” signs. The middle part reflects the attributes of the class and the bottom section the methods of the class. The dashed lines between the classes (rectangles) represent a dependency, while the solid lines represent an association with a description of the association.

Activity diagramming

The activity diagram's function is to model high-level business processes, or the transitions between states of a class.

Figure 4-9 shows an activity diagram. In the activity diagram, the filled circle at the top indicates the starting point of the activity diagram. The rounded rectangles represent processes, or an activity that is performed. The text on the arrows represents conditions that must be fulfilled. The diamond represents decision points. The thick bars represent the start and end of potentially parallel processes. The filled circle at the bottom represents an ending point.

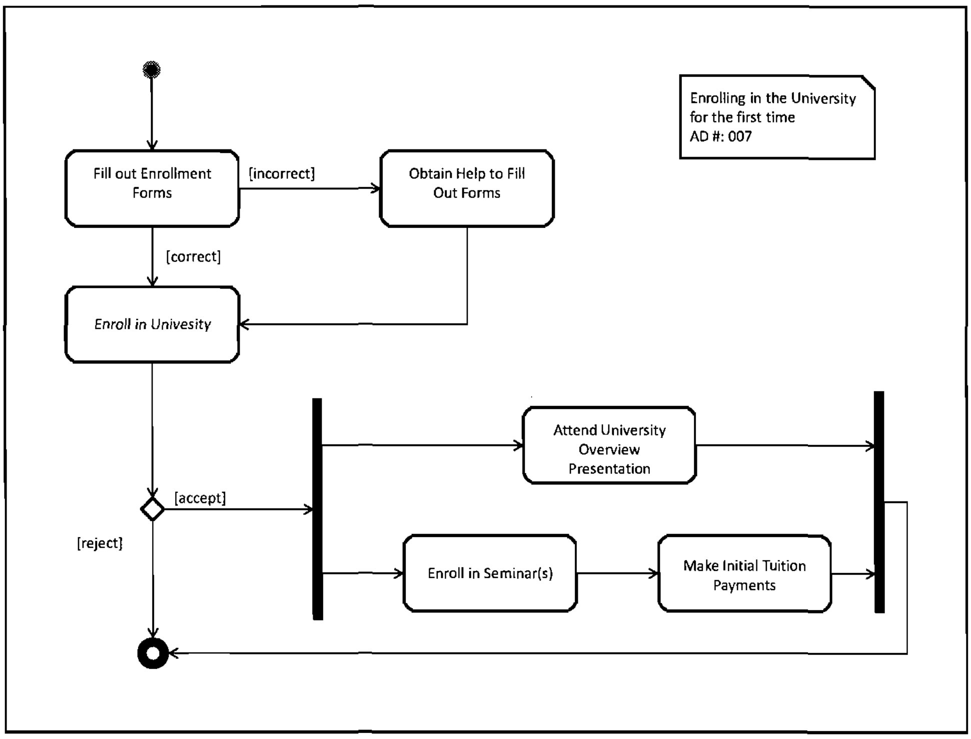

Enrolling in the University
for the first time
AD #: 007
Fill out Enrollment Forms
[incorrect]
Obtain Help to Fill Out Forms
[correct]
Enroll in Univesity
Attend University Overview Presentation
[accept]
[reject]
Enroll in Seminar(s)
Make Initial Tuition Payments

Figure 4-9 UML activity diagram (Ambler, 2001:230)

User interface prototyping

User interface prototyping is an activity in which users are actively involved in the making-up of the system's user interface. The purpose of this is to explore the problem space the system needs to address and to allow for the exploration of the solution space from the users' point of view. It also allows for a vehicle to communicate possible user interface designs.

The process is an iterative process consisting of the following steps:

- Determine the needs of the users.
- Build the prototype.
- Evaluate the prototype.

The above process is repeated until no further new ideas can be generated from the prototype.

Evolving the supplementary specification

During the analysis phase, one's understanding of the content of the supplementary specification evolves. It reflects mainly on the constraints, business rules and non-functional requirements identified during the requirements defining phase. It is likely that information originally specified, does not always contain enough detail and therefore should be detailed further.

User documentation

Due to the complexity of the systems, it is recommended that they should be well documented. It is imperative to furnish the following documentation:

- Tutorial manual
- Reference manual
- User manual
- Support user guide.

4.4.2.4. Object-Oriented Design (OOD)

Coad and Yourdon (1991:3) distinguish between object-oriented analysis (OOA) and object-oriented design (OOD) in object-oriented approach. OOA models the problem domain and the system's responsibilities. The OOD is an implementation of an OOA model (Coad & Yourdon, 1991:178).

Ambler (2001:250) explains that the function of modelling a design is to determine how the system should be built and to obtain the information required to drive the implementation of the system.

Before designing the system, the following should be considered: (Ambler, 2001:250):

- To design using a pure object-oriented solution or to design using a component based solution. An object-oriented solution is built from a collection of classes, while a component based solution is built from a collection of components. These components can be a non-object-oriented technology.
- To design using a common business architecture; the business architecture can be implemented straight through, or partially.
- Which non-functional requirements and constraints will be supported and to what extent.

The following discussion is based on Ambler (2001).

Layering models

Layering is to organise the software design into different collections of classes, or components that fulfil a common purpose. This increases the extensibility, maintainability and portability of systems created. Figure 4-10 illustrates the layering on class types.

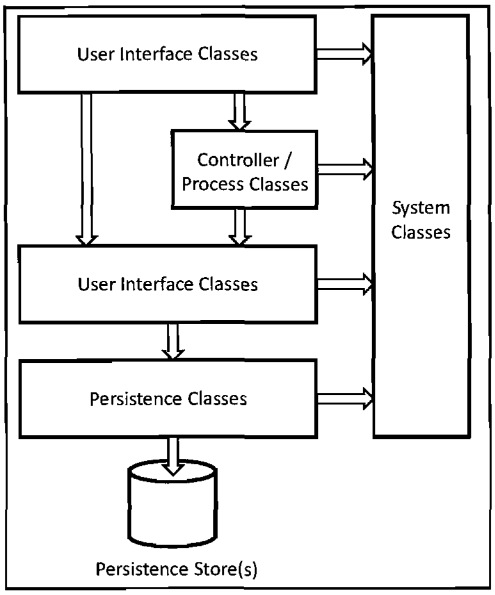


Figure 4-10 Layering system based on class types (Ambler, 2001:255)

From Figure 4-10, it is evident that the arrows all point downwards, illustrating that the flow of messages can only go in this direction.

The components are grouped as follows:

- User interface classes – These classes contains all the code needed for the graphics user interface to function.
- Controller/Process Classes – These classes implement the business logic.
- Business/Domain Classes – these classes encapsulate the basic business functionality.
- Persistence Classes – These classes provide the infrastructure to store and retrieve information.
- System Layer – This provides access to the operating system.

Class Modelling

The function of class modelling is to model the static structure according to which the software will be built. This structure typically focuses on the solution space and is more specific to the technical environment.

The following is modelled by using class modelling:

- Inheritance classes
- Association and dependency of classes
- Aggregation and composition of classes
- Attributes of classes

Rumbauch *et al.* (1991:168) rounds off class modelling by grouping the classes into modules, these sub-sets should capture some logical sub-sets of the entire model.

Inheritance classes

Rumbauch *et al.* (1991:163) explains that inheritance in classes share a common structure. This can be added in two directions: one is to generalise common aspects of existing classes into a super class (bottom-up), or to refine existing classes into specialised subclasses (top-down). To generalise common aspects, one needs to search for classes with similar attributes, associations, or operations, and define a super class that shares a common feature.

To refine existing classes into subclasses, a search for noun phrases composed of various adjectives on the class name should be conducted.

Association and dependency of classes

Rumbauch *et al.* (1991:31) explains that associations indicate the class sharing the information in subclasses. OO uses pointers to indicate associations.

According to Booch (1994:109), there are three types of cardinality in associations. These are:

- One to one
- One to many
- Many to many

An association also indicates the dependencies of classes on one another (Booch, 1994:109).

Aggregation and composition of classes

Booch (1994:128) states that aggregation relationships among classes have a direct parallel to aggregation relationships among the objects corresponding to these classes. This means that when two classes are coupled tightly together, the one class will always instantiate with the other class and cannot be dealt with independently. Rumbauch *et al.* (1991:36) explains this as a "part-whole" or "a-part-of" relationship between the components. Two kinds of aggregations are illustrated, aggregation as containment by value and aggregation as containment by reference.

Modelling attributes of classes

The attribute's name should indicate what it represents; this should be in the format of "attributeName".

The second factor of significance is the attribute's visibility. UML supports three types of attribute visibility, namely public, protected and private. Ambler suggests that all attributes in a class should be declared private, as this promotes information hiding.

Thirdly, all attributes should be documented for any developer to understand its purpose. The documentation should contain the description of the attribute, the applicable invariants, being the conditions under which the attribute is true, as well as the examples and visibility decisions explaining the reasons why the given attribute is declared as such.

The following approach should be followed when designing attributes:

- Assign private visibility to all attributes.
- Update an attribute only in its setter methods.
- Directly access an attribute only in its getter methods.
- Always invoke a setter method for an attribute to update its value.

- Always invoke a getter method for an attribute to obtain its value.
- Implement simple validation logic for an attribute in its setter method.
- Implement complex validation logic in separate methods.

Figure 4-11 is an example of the class design for two classes, i.e. student and studentNumber. The "+" sign means the attribute is public and the "-" means the attribute is private. A "#" means the attribute is protected (Ambler, 2001:284).

The arrow from the student to the studentNumber indicates the cardinality of the class.

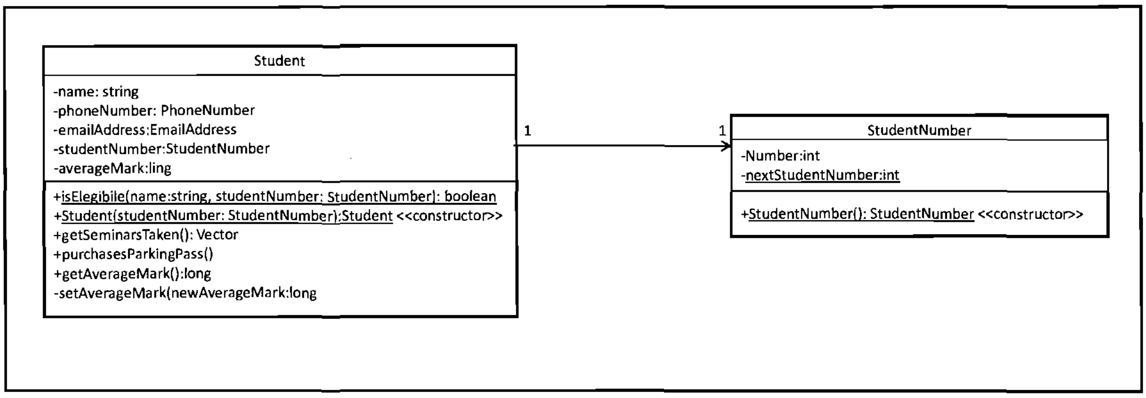

Student
-name: string
-phoneNumber: PhoneNumber
-emailAddress:EmailAddress
-studentNumber:StudentNumber
-averageMark:ling
+isElegibile(name:string, studentNumber: StudentNumber): boolean
+Student(studentNumber: StudentNumber):Student <<constructor>>
+getSeminarsTaken(): Vector
+purchasesParkingPass()
+getAverageMark():long
-setAverageMark(newAverageMark:long
1
1
StudentNumber
-Number:int
-nextStudentNumber:int
+StudentNumber(): StudentNumber <<constructor>>

Figure 4-11 The student and studentnumber design classes (Ambler, 2001:282)

State chart modelling

Avison and Fitzgerald (2003:253) explain that the function of a state chart diagram is to illustrate the various permitted states an object may be in.

Figure 4-12 is an example of a state chart diagram, also known as a state diagram. The state is a particular set of values of the attributes of an object at a particular time. When these values change, the state also changes.

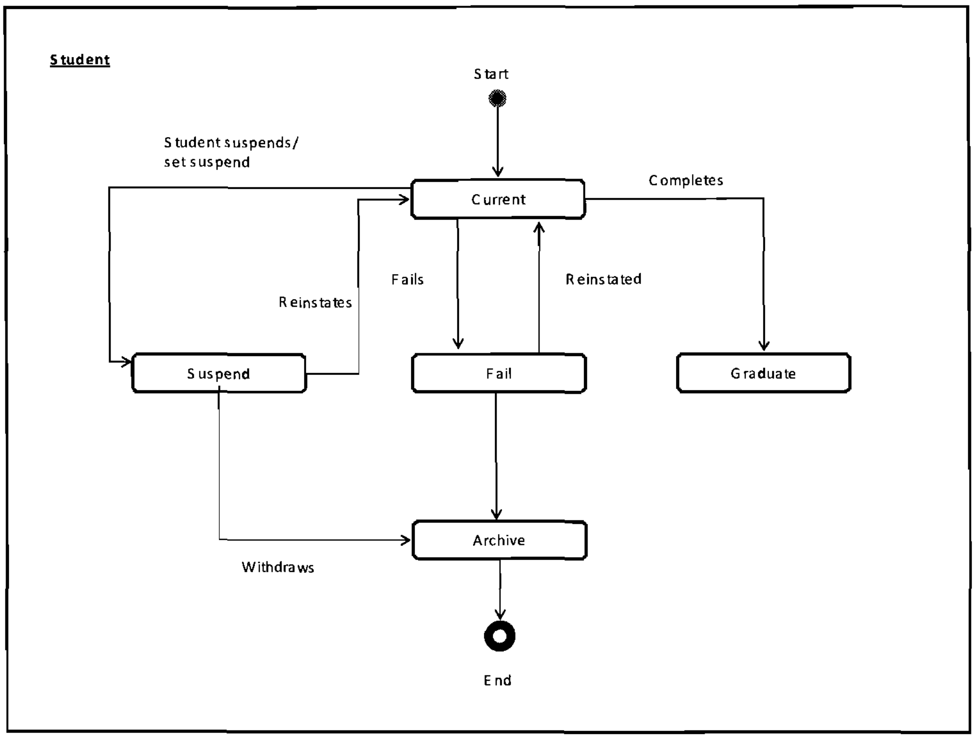


Figure 4-12 State chart diagram for student object (Avison & Fitzgerald, 2000:254)

The states of an object (like the student object example above) are represented by rectangles labelled with that state. Transitions are represented by the arrows associated with the name of the event that triggers the change.

The solid black dot represents the starting point and the bull's eye the end of the flow of states.

Collaboration modelling

Collaboration models provide a bird's eye view of the collection of collaborating objects. This model shows the message flow between objects in an OO application and also implies the basic association between the classes.

The rectangles represent the classes and the lines connecting the rectangles represent the association. The descriptions above the association lines are methods used to complete the association.

Figure 4-13 is an example of a collaboration diagram.

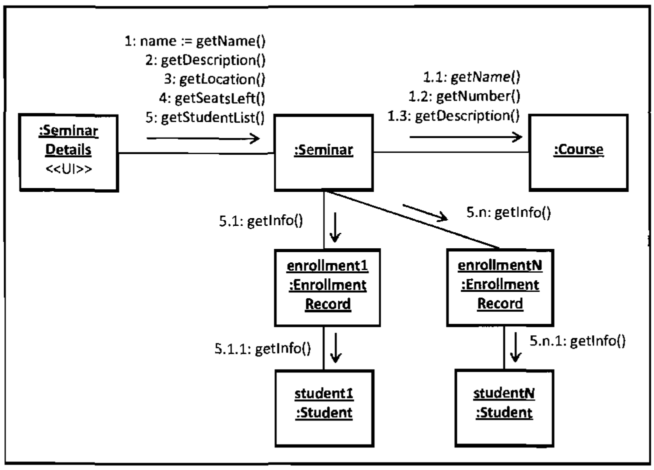


Figure 4-13 A collaboration diagram (Ambler, 2001:302)

Deployment modelling

Deployment modelling depicts a static view of the run-time configuration of processing nodes and the components running on those nodes. This diagram is essential in cases where the system is deployed onto several machines.

Figure 4-14 illustrates a UML deployment diagram. The three-dimensional boxes represent a node, which can be a computer or a switch. The connection is represented by a dotted line between the boxes.

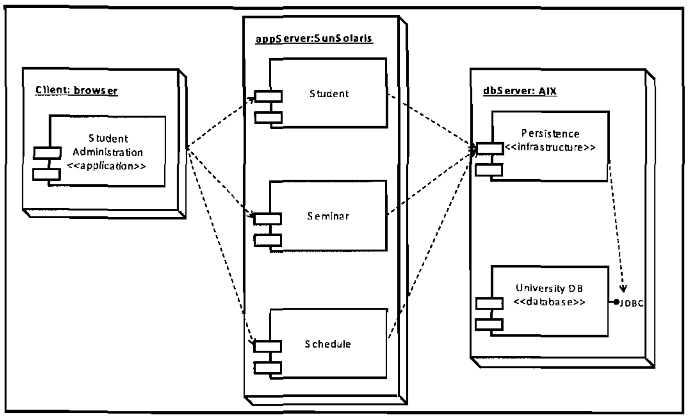

appServer:SunSolaris
Client: browser
Student Administration <<application>>
Student
Seminar
Schedule
dbServer: AIX
Persistence <<infrastructure>>
University DB <<database>>
JDBC

Figure 4-14 Deployment diagram (Ambler, 2001:313)

User interface design

User interface design concludes OOD. This activity is based on the use interface prototyping done during analysis.

The application of common user interface design principles and techniques is required. It is recommended that the following principles be kept in mind when designing interfaces:

- Structure – the interface should be designed purposefully in meaningful and useful ways to be clear, consistent and recognisable.
- Simple – the design should be simple to use.
- Visibility – options should be visible without any redundant distractions.
- Feedback – the users should be informed constantly of the actions of the system.
- Tolerance – the design should be flexible and tolerant to reduce the cost of mistakes and misuse.
- Re-use – the design should re-use external and internal components and behaviours to maintain consistency.

During the user interface design, the flow will also be modelled. This is done by using the interface flow diagram.

Once the design is finished, development of the solution may start. The artefacts required for the implementation, (Ambler, 2001:348), are the following:

- User interface prototype
- State chart diagram
- Class model
- Collaboration Diagram
- Business rules

4.4.3. Rational Unified Process (RUP)

Jacobson *et al.* (2001:4) explains that the rational unified process (RUP) is a use case driven, architecture-centric and iterative process that uses the unified modelling language (UML) to produce its blueprints for a software system.

The use case is used to capture the system's requirements and the combination of all use cases makes up the use case model. This describes the complete functionality of the system.

The use cases drive the system's architecture and the system's architecture in turn influences the selection of use cases. The maturity of these components is driven by the life cycle.

The architecture is cast in a so-called form. This form is based on the key use cases explaining the core functionality of the system. As the lifecycle continues, the architecture grows until it is deemed stable.

RUP is an incremental and iterative process. This means that the user requirements cannot be determined all at once. Every iteration identifies and specifies the relevant use cases.

The above-mentioned three concepts provide the structure according to which RUP works and are interdependent.

4.4.3.1. Life cycle of RUP

RUP's life cycle repeats a series of cycles that makes up the life cycle of the system. Every cycle is seen as a release, divided into the following phases:

- Inception
- Elaboration

- Construction
- Transition

Each phase has a workflow, defined as a sequence of activities producing a visible result (Avison & Fitzgerald, 2003:426). Due to RUP's popularity and flexibility, multiple variances in terms of the workflows are found. This discussion will be based on the original RUP (Jacobson *et al.*, 2001).

Every release is a product ready for delivery and includes the requirements, use cases, non-functional requirements and test cases (Jacobson *et al.*, 2001:9).

Figure 4-15 illustrates the core workflows, i.e. requirements, analysis, design, implementation and testing. These workflows take place over the four phases (inception, elaboration, construction and transition). The curves are approximations of the extent to which each workflow is carried out over a particular phase.

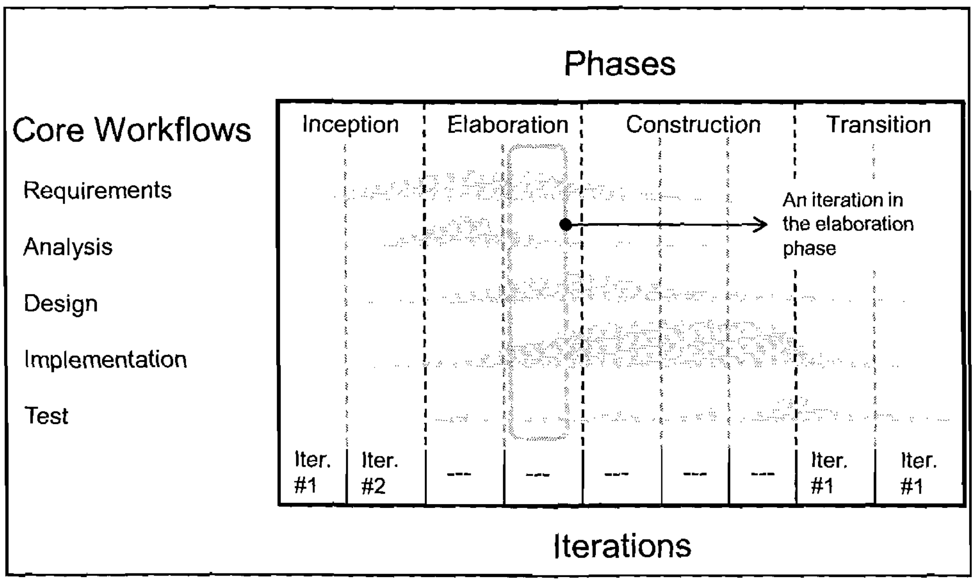


Figure 4-15 The five work flows that takes place over the four phases (Jacobson *et al.*, 2001:11)

Requirements workflow

The discussion on the requirements workflow is based on the work of Jacobson *et al.* (2001). It starts with the development of the business models. These techniques describe the business processes of the organisation. The result of the modelling is the domain model.

The activity is supported by two kinds of UML models:

- Business use case models being the same concept as essential use case models, described in OOSP.
- A business object model describing how each business use case is realised.

The domain model sets the context of the system, while the use case model captures the functional requirements and the individual use cases the non-functional requirements.

The use case model is described as a whole, a set of diagrams and a detailed description of each use case (similar to the essential use case description in OOSP).

User interface prototypes are produced for each actor representing the user interfaces. A supplementary requirements specification is created for generic requirements not specific to a particular use case.

Analysis workflow

The discussion on the analysis workflow is based on Jacobson *et al.* (2001).

The analysis workflow produces an analysis model, which is a conceptual model analysing the requirements through refinement and structure. The model includes the following:

- Analysis classes

- Use case realisations
- Analysis packages
- Service packages
- The architecture description

Analysis classes focus on the abstraction of classes, or subsystems in the design. They contain the following:

- Responsibilities
- Attributes
- Relationships
- Special requirements

The three existing types of classes are:

- Boundary class – models interaction between the system and the actors. It is involved in receiving and requesting information from external systems and users. This is similar to the user interface classes in OOSP.
- Control class – co-ordinates sequence, transacts and controls other objects. It is used to encapsulate control of a specific use case. This is similar to the business classes in OOSP.
- Entity class – models information that is persistent. It includes the information about the entity and its associated behaviour. This is similar to the actor classes in OOSP.

Use case realisation analysis collaborates within the analysis model. It describes how use cases are realised and performed in terms of the analysis classes. It provides a trace to a specific use case in the use case model. A typical artefact used for this, is a collaboration diagram. Analysis packages can be used to organise the artefacts of the analysis model into manageable pieces. These artefacts, which are in the use case realisation, are the analysis classes that are grouped together. The packages should be cohesive and loosely coupled. Services offered by the system, are grouped together by service packages.

The architecture description contains the architecture view of the analysis model. It furnishes a decomposition of the analysis model into its analysis packages and their dependencies. It also contains the key analysis classes, such as the entity-, boundary- and control classes. Furthermore, it contains the use case realisations that realise the important and critical functionality of the system.

The output of the analysis workflow will be used for the design workflow. The analysis- and service packages will impact on the design of the analysis- and service subsystems.

Analysis classes will be used as specifications for designing classes. Use case realisation analysis is the technique that will create more precise specifications for the use cases. It will also serve as an input to designing the design use cases.
The architecture view of the analysis model will serve as an input to the architecture view of the design model.

Design workflow

The discussion on the design workflow is based on Jacobson *et al.,* (2001). This workflow produces the designs that serve as a blueprint for the implementation of the system.

The processes executed, are design class, use case realisation design, design subsystem, interfaces, deployment diagram and a description of the architecture.

The design class follows the same method for defining classes, as previously discussed in OOSP, and also makes use of a class diagram. The use case realisation is the design process describing the events of a class diagram (called a system use case in OOSP) and an interaction diagram (called a sequence diagram in OOSP).

The subsystem process is a mean of organising the design model into manageable pieces. It consists of:

- Design classes
- Use case realisation – designs and interfaces.

As in the case of the analysis subsystem, it should be cohesive and loosely coupled. Service subsystems fulfil the same function as previously discussed for other service subsystems.

The user interface design is also done in this workflow and is executed in the same fashion as for OOSP. The architecture description contains the architecture view of the design model. It also contains a decomposition of the design model into the design packages and their dependencies. Lastly, it should contain the use case realisations; designs that need to be developed for the system. The key design classes trace back to the key analysis classes.

The deployment model is a model that describes the network configurations, nodes, the active mapping between the classes and the nodes and an architectural view of the deployment model.

The output of the design workflow will be used in the implementation workflow. The design- and service subsystems will be implemented by the implementation subsystems. The design classes will be implemented by the file components.

The use case realisation designs will be used by way of small steps that produce "builds". Lastly, the deployment model and the network configurations will be used to distribute the exceptionable components onto the nodes.

Implementation workflow

The discussion on the implementation workflow is based on Jacobson *et al.* (2001).

This workflow produces the implementation model required for the implementation of the design model. It also describes the organisation of the component and its structure, as well as the dependencies of the components in terms of the implementation environment.

The implementation workflow produces:

- Components
- Implementation subsystems
- Interfaces
- Implementations
- Architecture descriptions
- Integration "build" plans

A component is a physical package of model elements, the model elements being typically the design classes in the design model. Components can be one of the following stereo types:

- <<executable>> - indicates a program that can be executed.
- <<file>> - indicates a file that contains source code.
- <<library>> - indicates a static or dynamic library.
- <<table>> - indicates a database table.
- <<document>> - indicates a document.

Implementation subsystems provide a mean to organise the design artefacts into more manageable pieces. This can be a combination of components, interfaces and other subsystems.

Interfaces in the component should implement all the operations defined by the interface, and the dependencies supporting these operations should be available. Code is used to create these interfaces.

The architecture description of the implementation workflow contains the architecture view of the implementation model. It also contains a decomposition of the implementation model into the design packages and their dependencies. The key components trace back to the key design classes, executable components, and components central to other components dependent on them.

Every "build" is a step that is released and tested for integration and the system's test. Every "build" is version-controlled, thus enabling rollback in case of a faulty new release. Every "build" is subject to a standard of testing. The test workflow focuses on this.

Test workflow

The discussion on the test workflow is based on Jacobson *et al.* (2001).

This workflow produces a test model, consisting of:

- Test cases
- Test procedures
- Test plan
- Test evaluation
- Test components

Test cases specify a way of testing the system. This includes what need to be tested, with which inputs or results, and under which conditions. Test procedures specify how to perform one or several test cases, or parts thereof. This can be an instruction of how to test manually, or how to use an automated testing tool.

Test components automate one or more test procedures, or parts thereof. The test plan describes the testing strategies, resources and schedules. It also includes the type of tests to be performed and the objects for these tests.

A defect is produced when a system anomaly is found; this is tracked and resolved. Evaluation tests are the results of test efforts, such as test-cases coverage, code coverage and the status of defects.

4.4.4. Object Modelling Technique (OMT)

A literature study showed that Rumbaugh *et al.* (1991) are the original authors of the OMT methodology. The methodology was created before unified modelling language (UML) existed and uses its own notation to describe objects and classes. This discussion will follow the original notation.

The OMT methodology consists of three phases (Rumbaugh *et al.*,1991:145):

- Analysis
- System Design
- Object Design

4.4.4.1. Analysis

The following discussion is based on Rumbaugh *et al.* (1991).The analysis phase concentrates on the understanding and the modelling of the application and the problem domain. It starts with a problem statement. This is an information statement to be refined by the analyst at a later stage.

The requirements document typically contains the following:

- Problem scope
- What is needed
- Application context
- Assumptions
- Performance needs

Once the initial requirements are finalised, the analysis of the requirements can be done. The models created during the analysis, are the following:

- Object model
- Dynamic model
- Functional model

Object model

The object model models a static structure of the objects in the problem domain and organises this into workable pieces.

The information for this model comes from the problem statement and expert knowledge. The steps in object modelling are the following:

- Identify objects and classes – nouns in the problem domain are usually classes, while entities are usually objects.
- Prepare a data dictionary – a description of the objects, the scope of the class and the restriction on the object or class usage.
- Identify associations and aggregations between objects – usually correspond to verbs or verb phrases. These verb phrases should be documented first (from the problem statement).
- Identify attributes of objects and links – correspond to possessive nouns, e.g. “the collector of the car”. The adjective frequently represents the enumerated attribute value. The attributes do not necessarily come from the problem

statement, and one should make use of knowledge of the problem domain. A link attribute is a link between two objects, e.g. the many-to-many association of the objects stockholder and company, is a number of shares.

- Organise and simplify object classes with inheritance – done by searching for classes with similar attributes, associations or operations. It can be accomplished in two directions: generalising common aspects of existing classes into a super class, or refining existing classes into specialised subclasses.
- Verify that access paths exist for likely queries – done by testing access paths with meaningful questions. For example, what identifies a bank account, or can one access more than one bank account from an ATM.
- Iterate and refine the model – the first iteration is only a start. Models are identified and modified by each iteration.
- Group classes into modules – classes that are tightly coupled, need to be grouped together into modules. A module captures a logical subset of the entire model.

Figure 4-16 illustrates an example of an object model. The rectangles in Figure 4-16 represent the objects and classes. The name of the class is on top, with its attribute(s) listed below the name. The association between classes is depicted by a line between the classes. These associations are verbs and represent action taken.

Once the object model is defined, the dynamic model can be modelled.

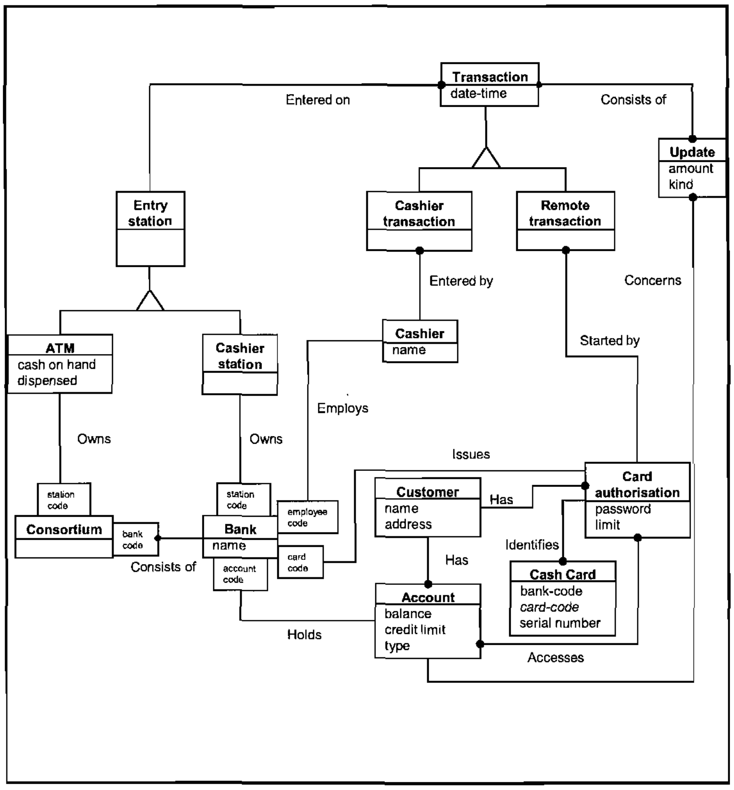


Figure 4-16 Object model (Rumbaugh *et al.*, 1991:168)

Dynamic model

The dynamic model models the time dependency behaviour of the system. This analysis searches for events and summarises the event sequence by using a state diagram.

Static data does not contain events, therefore no models are created for these systems. Dynamic models are more suitable for interactive systems. To construct a dynamic model, the following steps need to be performed:

- Prepare scenarios of typical interaction sequences – a scenario is a description of the sequence of events. This is similar to a use case scenario used in OOSP and RUP.
- Indentify events between objects – the use scenarios identify the events. An event is anything that needs interaction from the external world.

- Prepare an event trace for each scenario – a list of events among the objects. This is similar to a sequence diagram.
- Build a state diagram – this is similar to a state chart diagram used in OOSP and RUP.
- Match events between objects to verify consistency – by checking for completeness and consistency with the state diagram for all the objects.

Functional model

Functional models show how values are computed. This model does not take the sequencing and structure of objects into consideration. Data flow diagrams are used to show the functional dependencies. Functions are expressed in the form pseudo code, natural language, or mathematical equations.

The following steps are performed to create the functional model:

- Identify input and output values – the values needed for an event. The problem statement is a good source for defining inputs and outputs.
- Build data flow diagram showing functional dependencies – a diagram that illustrates which outputs connect to which inputs. The data flow diagram is created in layers, each detailing into deeper levels of the model.
- Describe functions – once the dataflow diagram is sufficiently refined, the function can be described by means of natural language, mathematical equations, pseudo code, or decision tables.
- Identify constraints – identify functional dependences between objects not related by an input-output dependency.
- Specify optimisation criteria – values that need to be maximised, minimised or optimised. For example, messages sent between different ATM sites.

Figure 4-17 is an example of a function description.

```
Update account (account, amount, transaction-kind) -> cash, receipt, message
    if the amount on a withdrawal exceeds the current account balance,
        reject the transaction and dispense no cash.
    if the amount on a withdrawal does not exceed the current amount balance,
        debit the account and dispense the amount requested
    if the transaction is a deposit
        credit the account and dispense no cash
    if the transaction is a status request
        dispense no cash
    In any case,
        the receipt shows ATM number, date, time, account number,
        transaction-kind, amount transacted (if any), and new balance
```

Figure 4-17 Example of a function description (Rumbaugh *et al.*, 1991:183)

The analysis phase is an iterative process. The model serves as a specification of the problem and the problem domain, without introducing an implementation.

The analysis document is the problem statement, object models, dynamic models and the functional models.

4.4.4.2. System Design

The following discussion is based on Rumbaugh *et al.* (1991).

Analysis is required for determining what needs to be done with regard to a problem of the system. Design is required for determining how the solution should be performed. During the system's design, the overall structure is decided. This forms the architecture, which is the organisation of the system into components known as subsystems. The architecture serves as the context within which the detailed decisions are made during the systems design. The decisions to be made are the following:

- Organise the system into subsystems – package of classes that represents a well defined service interfacing with the system. Each subsystem can be divided into smaller pieces known as modules.
- Identify concurrency inherited in the problem – not all software objects are concurrent. One needs to identify which objects can be used together and which need to be mutually exclusive.

- Allocate subsystems to processors and tasks – concurrent subsystems must be allocated to a hardware unit, which can be a general purpose processor, or a specialised functional unit.
- Choose an approach for management of data stores – this can be in the form of files, or in the form of database management systems (DBMS).
- Handle access to global resources – global resources, such as processors, tape drives, disks, etc. need to be identified. A lock is an object that handles the object. These locks need to be defined.
- Choose the implementation of control in software – analysis shows interactions as events among objects. The designer must decide on ways to implement control on these interactions. Two types of control exist:
  - External control – flow of externally visible events among the objects of a system.
  - Internal control – flow of control within a process.
- Handle boundary conditions – the designer must provide for the following three types of boundary conditions that exist:
  - Initialisation – the system must be brought from a quiescent initial state to a sustainable steady state.
  - Termination – the system object is abandoned.
  - Failure – the unplanned termination of a system.
- Set trade-off priorities – the design must provide for a decision to be made between desirable, but incompatible goals. These, for instance, may affect the performance of a development, or event, where a decision is required as to whether certain functions should be dropped in order to complete the development on time.

The system design document consists of the structure of the basic architecture of the system and the high level strategy decisions.

4.4.4.3. Object Design

The following discussion is based on Rumbaugh *et al.* (1991). The analysis phase determines what the implementation needs to fulfil, the system's design determines how this will be accomplished. The object design determines the full definition of classes and associations used in the implementation, as well as the interfaces and algorithms. During this phase, the analysis model is refined and detailed. The steps used in the object design phase, are:

- Combine the three models to obtain the operations and classes – the actions and activities of the dynamic model and the processes of the functional model are converted into operations attached to the classes in the object model. This is the mapping of the logical structure into a physical organisation of an application.
- Design the algorithms to implement operations – each operation specified in the functional model, must be formulated as an algorithm; the algorithm tells how a certain process should be done.
- Optimise access paths to data – the analysis model can be optimised for greater performance, the trade-off being between an optimised or less optimised model and a non-optimised generic model. For optimisation, the designer can apply one of the following to the analysis model:
  - Add redundant associations to minimise access cost and maximise convenience.
  - Re-arrange the computation for greater efficiency.
  - Save derived attributes to avoid recompilation of complicated expressions.
- Implement control for external interactions – the strategy for implementing the state-event models present in the dynamic model, must be refined. In the system design, a strategy is taken on the implementation control in the software. In the object design, the designer needs to flesh out this strategy by means of the following approaches:
  - Procedure-driven system approach – to use the location within the program to hold the state of the program.

  - Event-driven system approach – to use a direct implementation of a state machine mechanism, such as a state engine determining the next state of objects.
  - Control as current tasks – an object can be implemented as a task.
- Adjust class structure to increase inheritance – as the design progresses, more classes and operations appear. The designer can follow the under-mentioned approaches to increase the amount of inheritance in the system:
  - Re-arrange and adjust classes and operations.
  - Abstract common behaviour from classes.
  - Use delegation to share behaviour in cases where inheritance is semantically not valid.
- Design associations – provide access paths to objects. During object design, a strategy is formulated for implementing associations. The designer needs to analyse the associations to make use of the following association techniques:
  - One-way associations – an association that is only traverse in one direction.
  - Two-way associations – an association that is traverse in both directions.
  - Link attributes – are used for implementation of multiplicity associations of an object.
- Determine objects attribute representation – the designer must choose when to use primitive types in representing an object and when to combine groups of related objects.
- Package classes and association into modules – this involves the following tasks:
  - Information hiding – the goal is to treat classes as "black boxes" to which the interfaces are public but the internal details are hidden.
  - Coherence of entities – to organise entities such as classes, operations or modules to fit together towards a common goal.

- Constructing physical modules – the initial organisation of modules may not be optimal. Modules need to be defined in such a way that the interfaces are minimal and well defined.

The object design document consists of the detailed object models, detailed dynamic models and the detailed functional models. The implementation of the system is the translation of the analysis models into code.

Rumbaugh (1991:279) states that a non-object-oriented language can be used to implement the analysis model, but an object-oriented language greatly improves the concepts used in the analysis model.

From the methodologies discussed, it becomes evident that comparison of these methodologies is not exactly straight forward, the problem being that the dynamic classification framework is not a comparison framework, but a classification framework.

The next discussion explains a proposed framework that will be used to compare the methodologies discussed.

4.5. Comparing the ISD Methodologies

So far, this chapter concentrated on describing the OOA, OOSP, RUP and OMT methodologies. The following discussion will focus on comparing these methodologies in an attempt to find commonalities among them.

A literature study indicated that there are several techniques for comparing methodologies. In this discussion however, the technique described by Avison and Fitsgerald (2003:555) will be used. Figure 4-18 outlines the comparative review framework.

1. Philosophy
    a. Paradigm
    b. Objectives
    c. Domain
    d. Target
2. Model
3. Techniques and tools
4. Scope
5. Outputs
6. Practice
    a. Background
    b. User base
    c. Participants
7. Products

Figure 4-18 Outline for the comparative review of methodologies (Avison & Fitzgerald, 2003:556)

4.5.1. Philosophy

The first element discussed in the comparison framework, is philosophy. Avison and Fitzgerald (2003:557) explain that a philosophy is a set of principles underlying the methodologies.

From the methodologies discussed, one can derive that all methodologies use object-oriented concepts in an iterative fashion. The methodologies work in phases of the development, with some methodologies fo cusing on one phase only (OOA).

The components of the philosophy are paradigm, objectives, domain and target.

4.5.1.1. Paradigm

Avison and Fitzgerald (2003:557) identify two paradigms of relevance:

- The science paradigm – consists of reductionism, repeatability and refutation. This implies that the breakdown of the problem domain into parts does not disrupt the system of which it is a part.
- The systems paradigm – is a holistic approach concerned with the whole picture, the emergent properties and the interrelationships between parts of the whole.

Both hard and soft systems approaches are usually associated with this paradigm.

The paradigm of OOA follows a "divide and conquer" approach to solve the problem. This is a form of reductionism. The same argument can be applied to OOSP, RUP and OMT.

All of these methodologies are iterative in their development approach, which is a form of repeatability.

The phases of each methodology concentrate on a specific goal, e.g. the analysis phase found in OOA, OOSP, RUP and OMT concentrates on modelling the problem domain in a platform-free environment. This eliminates the restrictions posed by a platform. Elimination is a form of refutation.

The OOA, OOSP, RUP and OMT methodologies concentrate on the problem given and break it down into parts. These parts are modelled in terms of objects.

From the discussion above, one can derive that the OOA, OOSP, RUP and OMT models are all based on the scientific paradigm.

4.5.1.2. Objectives

The second component of philosophy is the objectives of the methodology. Avison and Fitzgerald (2003:557) explain that the objective of some methodologies is to develop a computerised system, while for others the objective is to determine whether an information system is really needed. This component determines the boundaries of the area of concern.

The objective of the Object-Oriented Analysis (OOA) methodology is to concentrate on the analysis of the problem domain and to model the problem domain in terms of objects. This methodology does not give any guidance as to gathering the requirements, designing the system, or implementing the system. It only concentrates on the analysis of the system.

The Object Modelling Technique (OMT) is a methodology that concentrates on more aspects in the development lifecycle. It includes the analysis phase and the designing phase, which are the system design and object design phases. However, it does not give any guidance on requirements gathering, other than a problem statement. There are no testing phases built into this methodology, and implementation is based on common object-oriented concepts.

The Object-Oriented Software Process (OOSP) is a methodology concentrating on requirements gathering, requirements validation, followed by the analysis of the requirements, systems design and the implementation of the system. The methodology does not include a testing phase, but makes use of another methodology, namely Full Life Object-Oriented Testing (FLOOT), to complete its testing.

The Rational Unified Process (RUP) is a methodology focusing on requirements gathering, analysis, design, implementation and testing. This methodology covers all areas of developing a solution.

All of the methodologies concentrate on developing a computerised system.

4.5.1.3. Domain

Avison and Fitzgerald (2003:560) state that systems and problems interrelate and that the solution to a number of interrelated problems is different to the sum of the solutions to the individual problems, viewed in isolation. They state that this led to a number of methodologies adopting a different philosophy. It caused methodologies to take on a much wider view of their starting point and is not looking to solve particular problems. They argue that, in view of the above for solving individual problems, it is necessary to analyse the organisation as a whole, devise an overall IS strategy, sort out the data and resources of the organisation and identify the overlapping areas and the areas that need to be integrated.

OOA, OOSP, RUP and OMT are methodologies specific to problem solving. They do not focus on identifying the systems required, but assume that a specific problem needs to be addressed.

4.5.1.4. Target

The last component in philosophy is the target. Avison and Fitzgerald (2003:560) explain that target focuses on the applicability of the methodology. Methodologies are targeted at particular types of problems, environment, type, or size of an organisation.

OOA concentrates only on analysing the requirements, thus solving the analysis phase. OOSP, RUP and OMT focus on a software solution aimed at fulfilling a business requirement.

4.5.2. Model

The second element in the framework is concerned with the analysis of the model the methodology adheres to (Avison & Fitzgerald, 2003:561). This is the basis of the methodological view, and can be seen as an abstraction and a representation of the important factors of the information system. There are four distinct types of models, i.e. verbal, analytic or mathematical, pictorial and simulation models.

All of the methodologies use diagrams to communicate designs. It can therefore be classified as pictorial models.

4.5.3. Techniques and tools

Techniques and tools are those used in a particular methodology (Avison & Fitzgerald (2003:561).

The techniques and tools used in OOA, are mainly class, object and gen-spec structure.

OOSP uses essential use case models, change cases, essential use interface prototypes, CRC models for domain modelling, user interface flow diagramming, user interface prototypes, use case models, sequence diagram, class model (analysis), activity diagrams, state chart diagrams, component diagrams, deployment diagrams, collaboration diagrams and class models (design).

RUP uses use case models, use case realisations analysis and design, interface analysis and design, architectural view of the analysis and designs, analysis and design classes, analysis and design packages, design subsystems, deployment models, test cases, test procedures and test components.

OMT uses problem statements, object models, dynamic models and function models. From these, an investigation is carried out on the system, and a systems design document is produced. The object design phase is concerned with adding platform considerations to the object design document.

4.5.4. Scope

The scope of a methodology is an indication of the stages in the life cycle (Avison & Fitzgerald, 2003:561). The scope of OOA is the analysis phase.

OOSP gathers the requirements, validates the requirements, analyses the requirements, designs the analysed requirements and implements the designs. The methodology follows a spiral life cycle approach.

RUP gathers the requirements, analyses the requirements, designs the analysed requirements, implements the designs and tests the implementation. This methodology also uses a spiral life cycle approach.

OMT gathers the requirements, analyses the requirements, designs the analysed requirements and implements the designs. It also uses a spiral life cycle approach.

4.5.5. Output

This component is concerned with the deliverables of the methodology at each stage and most importantly, the final deliverable (Avison & Fitzgerald, 2003:562).

In terms of output, the OOA methodology produces an analysed class and object model.

OOSP's requirements phase produces the functional and non-functional requirements for the problem domain. The validation phase produces change cases, if needed. The analysis phase produces an analysis model, used for the designs. The design phase produces the design model, used for the implementation phase.

RUP follows a similar output. The requirements mode is used for the analysis model, which in turn is used for the design mode and thereafter for the implementation model. The result of the implementation is tested in the test model.

OMT also produces the same output as RUP and OOSP. The analysis phase produces an analysis document, the systems design phase produces a systems design document, and these documents are used to produce an object design document. The object design document is used for the implementation phase.

4.5.6. Practice

The practice component in the framework is measured according to the following (Avison & Fitzgerald, 2003:562):

- The methodology's background in commercial or academic terms.
- The user base (not part of the discussion).

- The participants in the methodology and the required skills level. The practice should also include an assessment of difficulties and problems encountered, as well as perceptions of success and failure.
- It should be done by the users of the methodology. The results will be subjective, depending on who is consulted.

The background for OOA, OOSP, RUP and OMT is mainly based on object-oriented principles that have evolved into a full life cycle.

The OOA methodology is a methodology (Coad & Yourdon, 1991) that focuses on one aspect only. The principals used in OOA, are imbedded in OOSP, RUP and OMT in the analysis phase.

OOSP, RUP and OMT cater for a development full life cycle. The user base for OOA is the designers. The output, is utilised by them. The OOSP, RUP and OMT user base caters for architects, analysts, designers, programmers and testers, responsible for producing a full working software solution.

The participants in OOA are mainly the analysts, whereas in OOSP and RUP, the user base and business are included.

4.5.7. Product

Product is the last component in the framework, and as the name implies, this is the final deliverable of the methodology. It can range from a software product to a telephone help service (Avison & Fitzgerald, 2003:562). The product falls outside the scope of this study and will not be discussed.

Table 4-2 is a summary of the discussion on the comparative framework of Avison and Fitzgerald (2003).

	OOA	OOSP	RUP	OMT
Philosophy	Iterative, incremental, object oriented	Iterative, incremental, object oriented	Iterative, incremental, object oriented	Iterative, incremental, object oriented
Paradigm	Scientific	Scientific	Scientific	Scientific
Objectives	To analyse the requirements of the system.	To determine the requirements needed for a system, analyse the requirements, design the system and build it.	To determine the requirements needed for a system, analyse the requirements, design the system, build it and test the system.	To determine the requirements needed for a system, analyse the requirements, design the system, build the system.
Domain	Only the analysis domain	Only on the software solution domain.	Only on the software solution domain.	Only on the software solution domain.
Target	Analysis of the problem.	Software solution of the business problem domain.	Software solution of the business problem domain.	Software solution of the business problem domain.
Model	Pictorial	Pictorial	Pictorial	Pictorial
Techniques and Tools	Class and object diagrams and gen-spec diagrams	Essential use case models, change cases, essential use interface prototypes, CRC models for domain modelling, user interface flow diagramming, user interface prototypes, use case models, sequence diagram, class model (analysis), activity diagrams, state chart diagrams, component diagrams, deployment diagrams, collaboration diagrams and class models (design)	Use case models, use case realisations analysis and design, interface analysis and design, architectural view of the analysis and designs, analysis and design classes, analysis and design packages, design subsystems, deployment models, test cases, test procedures and test components	Problem statements, object models, dynamic models and function models from this an investigation is made on the system and a systems design document is produced. The object design phase concerns itself with adding platform considerations to the object design document.

Table 4-1 Comparison of the philosophies of the OO methodologies.

Scope	Analysis requirements	Gather requirements, validate requirements, analyse requirements, design analysed requirements, lastly implement designs, spiral life cycle	Gather requirements, analyse requirements, design analysed requirements, implements designs, tests implementation spiral life cycle.	Gathers the requirements, analyses the requirements, design the analysed requirements and implements the designs, spiral life cycle.
	OOA	**OOSP**	**RUP**	**OMT**
Outputs	Analysed class and object model	Requirements phase produces the functional and non functional requirements, change cases in validation, analysis models for design and design models for implementation	Requirements model, analyse model, design model, implementation model and test model.	The analysis phase produces an analysis document, the systems design phase produces a systems design document and these documents are used to produce an object design document. The object design document is used for the implementation phase.
Practice (Background)	Based on OO principles, caters for analysis	Based on OO caters for a full development life cycle.	Based on OO caters for a full development life cycle.	Based on OO caters for a full development life cycle.
Practice (User base)	N/A	N/A	N/A	N/A
Practice (Participants)	Analysts	Architects, analysts, designers, programmer, testers and business	Architects, analysts, designers, programmer , testers and business	Architects, analysts, designers.
Product	N/A	N/A	N/A	N/A

Table 4-2 (Continued) Comparison of the philosophies of the OO methodologies.

4.6. The general aspects of the comparison

In the previous paragraph, OOA, OOSP, RUP and OMT were compared to one another. The goal was to find commonalities among these methodologies.

From a philosophical perspective, the OOA, OOSP, RUP and OMT methodologies use object to explain reality.

The scope of OOA is to analyse the requirements, whereas in OOSP, RUP and OMT, the analysis phase is one of several phases.

All of these methodologies communicate by means of diagrams. The common phases highlighted, are:

- Requirements gathering – All methodologies cover a part of requirements gathering, except OOA, which assumes that requirements have already been dealt with and consequently starts with the analysis thereof.
- Requirements validating – highlighted only in OOSP, but of great importance to get a clear understanding of the problem.
- Analysis – All methodologies focus on analysing the problem.
- Design – All methodologies (except OOA) focus on design based on the analysis.
- Implementation – All methodologies (except OOA) focus on implementing the designs.
- Testing – All methodologies (except OOA and OOSP) focus on testing. OOSP uses a different methodology to test the implementation.

4.6.1. Requirements gathering

The main tool used for requirements gathering, is based on use cases. The use case diagram highlights the interaction between the actors and the processes (users/systems). Each relationship between the actor and the process is a use case.

For every relationship there must be an interface, the mean of linking the actor and process. From this, a domain is modelled on which requirements are based. The business has rules and processes, which are captured by using supplementary documentation.

Figure 4-19 represents the different activities in the requirements phase. The requirements phase is subdivided into two, the requirements gathering phase and the requirements validating phase.

Phase		Activity
Requirements	Gathering Requirements	• Essential Use Case Diagram • Essential Use Case • User Interface Prototyping • Domain Modelling • Supplementary Documentation
	Validating Requirements	• Use Case Scenario Testing • User Interface Walkthrough • Requirements Review

Figure 4-19 The requirements phase

4.6.2. Analysis

The analysis phase uses as input, the artefacts produced by the requirements phase. It starts with a more detailed use case, called a systems use case in OOSP and a use case realisation – analysis in RUP. OMT uses a functional description, which is a form of a use case.

From the above, a sequence model is built, specifying the interactions among the classes. A class model is built, while most of the concepts of OOA are used in OOSP, RUP and OMT.

The class model provides the detail that will be contained in the class. An activity diagram is created to explain the operation of a use case or method. The user interfaces captured in the requirements, are evolved into more specific detail. Next, the supplementary documentation and the user documentation are evolved and lastly, all components are organised into more manageable packages

All of these diagrams are platform non-specific. Figure 4-20 indicates the activities included in the analysis phase.

Phase	Activity
Object Oriented Analysis	• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages

Figure 4-20 The analysis phase

4.6.3. Design

The design phase aims at being more platform-specific. The class model is designed according to the rules of the programming platform, while a state chart diagram is used to illustrate the state of the object.

From the above, a collaboration diagram is created to trace the relationship between the class diagram and the sequence diagram. Component modelling is done to group the use cases, interfaces and class diagrams together.

A deployment diagram is created to illustrate the dependencies, as well as to specify the platform specifics.

Rational persistence modelling is done to model the persistence layer of the system. Lastly, the user interface is designed by making use of the platform specifics.

Figure 4-21 illustrates the design phase.

Phase	Activity
Object Oriented Design	• Class Modelling • State Chart Modelling • Collaboration Modelling • Component Modelling • Deployment Modelling • Relational Persistence Modelling • User Interface Design

Figure 4-21 The design phase

4.6.4. Implementation

The implementation phase focuses on the actual code development in accordance with the design phase. In this phase, all the classes are created according to the class design. The logic of the state charts and the user interfaces are coded.

The implementation is packaged according to the component design. The deployment environment is set up and the packages deployed. Figure 4-22 illustrates the implementation phase.

Phase	Activity
Object Oriented Implementation	• Code development • Component packaging • Deploy packages

Figure 4-22 The implementation phase

4.6.5. Testing

The testing phase tests the implementation in various ways and is completed by unit-testing every component. Figure 4-23 illustrates the testing phase.

Phase	Activity
Object Oriented Testing	• Unit test components

Figure 4-23 The testing phase

4.7. Summary

This chapter dealt with the detail of object-oriented systems development methodologies. It explained that the goal of OO is to ensure that:

- products are delivered to the user on time and within budget
- products meet user requirements
- increasingly sophisticated products are offered to keep a competitive edge
- the changes in standards and delivery technology are kept up
- the project team feels motivated and successful

The guiding principles and beliefs of the object-oriented development process is a seamless analysis, design and implementation process. The fundamental concepts of object-oriented development discussed, are:

- Problem domain vs. implementation domain
- Object and class
- Encapsulation
- Inheritance
- Polymorphism
- Communication among objects.

The lifecycle approach of object-oriented development is found to be iterative and incremental in nature.

The chapter also included a detailed discussion on four object-oriented development methodologies:

- Object Oriented Analysis (OOA)
- Object Oriented Software Process (OOSP)
- Rational Unified Process (RUP)
- Object Modelling Technique (OMT)

The second part of the chapter compared the four methodologies by using the comparison framework of Avison and Fitzgerald (2003).

The framework focuses on various aspects of a systems development methodology, such as:

- Philosophy
- Model
- Technique and tools
- Scope
- Output
- Practice
- Product

A summary of the discussion on the four methodologies and the framework is outlined in Figure 4-24. The figure represents the following six phases commonly found in object-oriented development:

- Requirements gathering
- Requirements validating
- Analysis
- Design
- Implementation
- Testing

Phase		Activity
Requirements	Gathering Requirements	• Essential Use Case Diagram • Essential Use Case • User Interface Prototyping • Domain Modelling • Supplementary Documentation
	Validating Requirements	• Use Case Scenario Testing • User Interface Walkthrough • Requirements Review
Object Oriented Analysis		• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages
Object Oriented Design		• Class Modelling • State Chart Modelling • Collaboration Modelling • Component Modelling • Deployment Modelling • Relational Persistence Modelling • User Interface Design
Object Oriented Implementation		• Code development • Component packaging • Deploy packages
Object Oriented Testing		• Unit test components

Figure 4-24 Summary of requirements, analysis, design, implementation and testing.

The next chapter will focus on the common data warehouse methodologies, and the matrix created in this chapter will be mapped to these methodologies.

Chapter 5 - Data Warehouse Development

5.1. Introduction

The purpose of this chapter is to focus attention on the data warehouse (DW) and its development methodologies.

Ramakrishnan and Gehrke (2003:848) explain that organisational decision making requires a comprehensive view of all aspects in the enterprise. For this reason, many organisations create data warehouses that contain data drawn from operational databases by difference business units, together with historical and summary information.

The previous chapter focused on object-oriented approaches and methodologies which are used in the development of operational systems. Little literature exists on object oriented data warehousing. This chapter introduces data warehousing as an addition to operational systems.

This chapter will focus on the following:

- Data warehouse development as a decision support system
- Differences between operational database systems and the data warehouse
- Data warehouse methodologies

5.2. The data warehouse

It is necessary to understand that the data warehouse serves a different purpose than the operational database system in the organisation.

Ramakrishnan and Gehrke (2003:847) explain that operational database systems maintain operational data. Operational data represents the day to day operations of the business.

Database systems typically execute small changes to data with a large number of transactions. They are optimised to perform more effectively for specific applications. Ramakrishnan and Gehrke (2003:847) further explain that there was a need in organisations to analyse the data. This is done through a data warehouse.

A typical data warehouse contains historical data that is analysed and explored. It identifies useful trends and creates summaries of data to support high level decision making. A data warehouse is also referred to as a type of Decision Support System (DSS).

The characteristics of data in data warehouses compared to operational database systems, also differ (Rob & Coronel, 2002:624). Table 5-1 lists the differences in characteristics of data in a data warehouse and data in an operational database.

Characteristic	Operational database data	Data warehouse data
Integrated	Similar data can have different representations or meanings	Provide a unified view of all data elements with a common definition and representation for all business units.
Subject-oriented	Data are stored with a functional, or process orientation.	Data are stored with a subject-orientation that facilitates multiple views of the data and facilitates decision making.
Time-variant	Data are recorded as current transactions.	Data are stored with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.
Characteristic	**Operational database data**	**Data warehouse data**
Non-volatile	Data updates are frequent and common.	Data cannot be changed. Data are added only periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.

Table 5-1 (Continued) A comparison of data warehouse and operation database characteristics (Rob & Coronel, 2002:624)

The design model of a data warehouse is also unique; it makes use of the so-called star schema design. Rob and Coronel (2002:641) explain that the star schema design allows the dimensional model to be mapped to a relational database.

The star schema comprises of the following components:

- Facts – numeric measurements that represent a specific aspect of an activity.
- Dimensions – characteristics that provide additional perspectives to a given fact.
- Attributes – descriptions that are found in a dimension.
- Attribute hierarchies – provide a top-down data organisation; this is used for drill-down / roll-up data analysis.

The fact table relates to each dimension in a many-to-one relationship.

Figure 5-1 is a representation of the sales fact and illustrates the different dimensions (location, customer, time and product) and the fact (sales). Each entity can have different record totals.

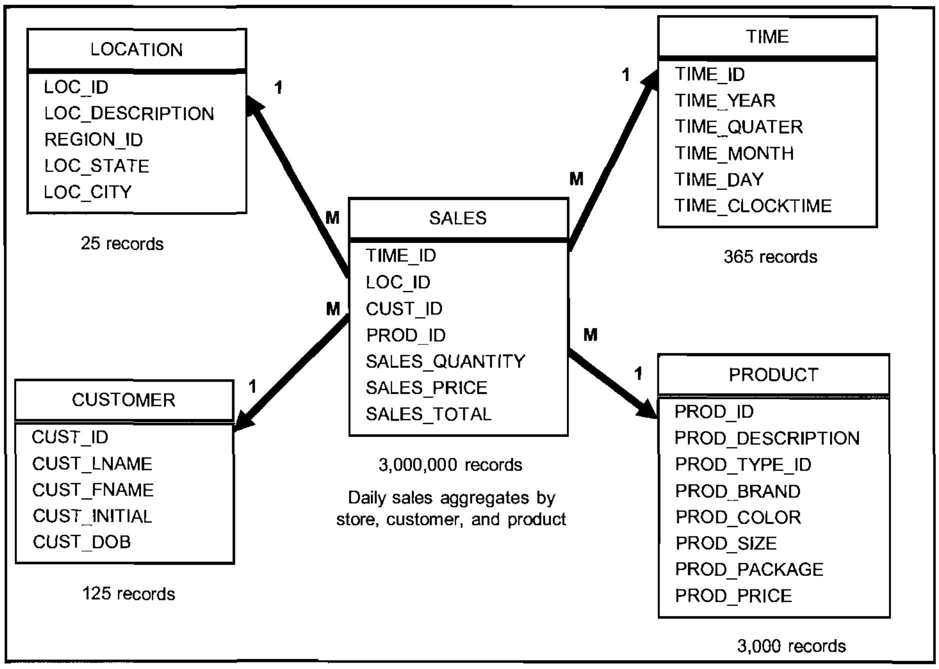


Figure 5-1 Star schema for sales (Rob & Coronel, 2002:647)

Advanced analysis tools utilise the data warehouse for their analysis processes. Ramakrishnan and Gehrke (2003:847) explain that one can find three classes of analysis tools:

- Online analytic processing (OLAP) – these tools typically support group by aggregation of data and complex boolean condition queries.
- Traditional structured query language (SQL) – some database management systems (DBMS) have built-in support for SQL queries that utilise OLAP performance.
- Trends and patterns search – exploratory data analysis and data mining are used on top of the data warehouse to find interesting patterns and behaviours in the data.

Typical data warehouse architecture will follow the concepts explained in Figure 5-2.

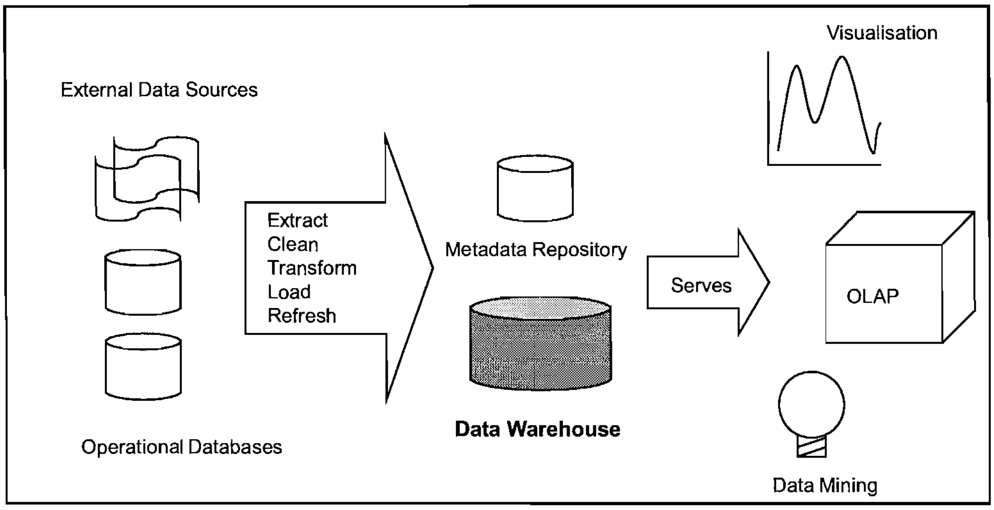


Figure 5-2 A Typical Data Warehouse Architecture (Ramakrishnan & Gehrke, 2003:870)

Figure 5-2 illustrates the sources (the operational databases) to the left. These sources are extracted, cleaned, transformed and loaded through an interface to the data warehouse. To the right are the reporting tools served by the data warehouse. The data warehouse is situated between the source systems and the reporting tools. From the above, one can derive that the data warehouse serves as a decision support system. This is a unique system compared to operational systems, where only the operational aspect of the business is supported.

The development methodology of data warehouses differs from that of traditional operational systems. The following discussion will focus on the different methodologies used in developing data warehouse solutions.

5.3. The data warehouse development methodologies

A literature study by Sen and Sinha (2005) found that the most recognised methodologies used in data warehouse development are those of Kimball *et al.* (1998) and Inmon (1996).

The methodology of Kimball *et al.* (1998) will be discussed before that of Inmon (1996) since it is the more important to the main argument of this dissertation.

5.3.1. The business dimensional lifecycle approach

Kimball *et al.* (1998:19) defines the data warehouse as “nothing more than the union of all constituent data marts. A data warehouse is fed from the data staging area “and a “queryable source of data in the enterprise”.

To fully understand the above definition, one should investigate the following:

- Data Mart – Kimball *et al.* (1998:18) states that this can be seen as a local subset of the complete data warehouse. A data mart is defined as a restriction of the data warehouse to a single business process, or to a group of related business processes targeted toward a particular business group.
- Data staging area – Kimball *et al.* (1998:16) defines this as “a storage area and a set of processes that clean, transform, combine, de-duplicate, household, archive and prepare source data for use in the data warehouse.” This is everything between the source system and the presentation server.

Kimball *et al.* (1998) uses the business dimensional lifecycle approach to develop data warehouses. This is illustrated in Figure 5-3.

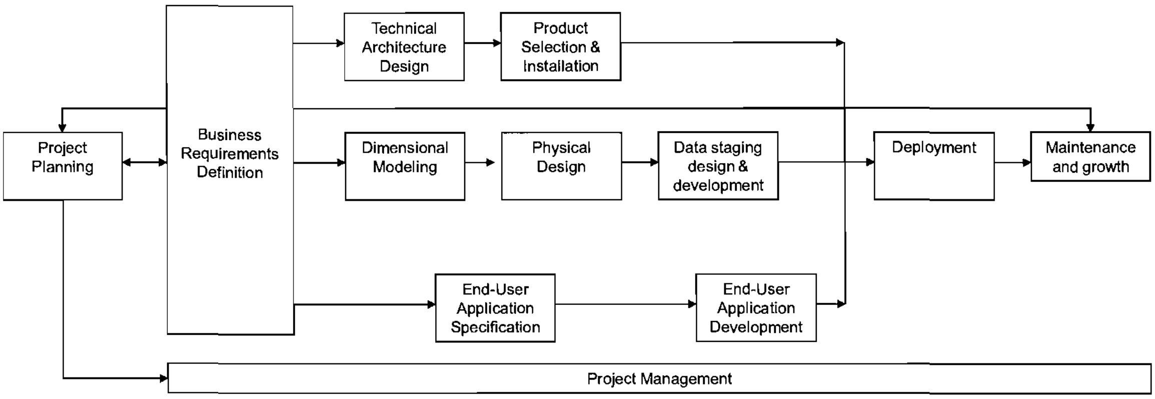

Technical Architecture Design
Product Selection & Installation
Project Planning
Business Requirements Definition
Dimensional Modeling
Physical Design
Data staging design & development
Deployment
Maintenance and growth
End-User Application Specification
End-User Application Development
Project Management

Figure 5-3 The business dimensional lifecycle (Kimball *et al.*, 1998:33)

The business dimensional lifecycle is a sequence of high level tasks required for effective data warehouse design, development and deployment. Three tracks, namely data-, technology- and application tracks are executed in parallel.

5.3.1.1. Project Planning

This phase addresses the definition and scoping of the data warehouse project, including readiness assessment and business justification. Kimball *et al.* (1998:33) states that project planning focuses on resource- and skill-level staffing requirements, linked with task assignments, duration and sequencing. The resulting integrated project plan identifies all tasks associated with the "Business Dimensional Lifecycle" and indicates the parties involved. It serves as the corner stone for the ongoing management of a data warehouse project. Project planning is dependent on the business requirements. This dependency is illustrated by the two-way arrow between project planning and business requirements in Figure 5-3.

The rest of the discussion on projection planning is based on the work of Kimball *et al.* (1998). To gain a more in depth understanding of the authors' methodology, one should examine the following aspects:

- Project definition and scoping
- Project planning

Project definition and scoping

Before a data warehouse or a data mart project is commenced, one has ensure that there is a demand and where the demand is coming from. If there are no strong business sponsor(s), and/or eager users, the project should be postponed.

The following five factors should be present before detailed work on the design and development of the data warehouse begins:

- Strong business management sponsor
- Compelling business motivation
- Is/business partnership
- Current analytic culture
- Feasibility

It is recommended that after the above assessment has been done, a preliminary scope should be developed. This preliminary scope should be based on business requirements.

Another aspect of project planning is business justification, which includes the following tasks:

- Determine the financial investments and costs
- Determine the financial returns and benefits
- Combination of the investments and returns, to calculate return on investment.

Project planning

Once the project is defined and approved, planning of the project may commence.

This entails the following:

- Establishment of project identity – giving the project a name.
- Staffing of the project – a data warehouse project requires a number of different roles and skills from both the business and IS communities. The authors illustrate the various roles by comparing them to a professional sports team:
    - "Front office: sponsors and drivers"
    - "Coaches: project managers and leads" – this consists of the following:
        - The project manager
        - Business project lead
    - "Regular project lead" – this consists of the following:
        - Business systems analyst
        - Data modeller
        - Data warehouse database administrator
        - Data staging systems designer
        - End user application developers.
        - Data warehouse educator
    - "Special teams" - these members contribute on a very specialised basis, including the following:
        - Technical/security architect
        - Technical support specialists
        - Data staging programmer
        - Data steward
        - Data warehouse quality assurance analyst
- Developing the project plan – two key words should be kept in mind when describing a project plan, namely 'integrated' and 'detailed'. The reason for this is that most data warehouse teams have multiple project plans that do not tie up.

5.3.1.2. Business Requirement Definition

The business requirements definition is the next phase in the business dimensional lifecycle approach. According to Kimball *et al.* (1998:95), this step is regarded as the centre of the "data warehouse universe". Figure 5-4 illustrates that business requirements have an impact on every aspect of the data warehouse project. To determine the business requirements, one should start by interviewing business users. A series of interviews must be conducted with each level of management in the organisation. The next step is to determine the success criteria, after which the preparation and publishing of deliverables follow. On completion of the above, the IS- and business teams should reach an agreement on the scope of the project (Kimball *et al.*, 1998:132).

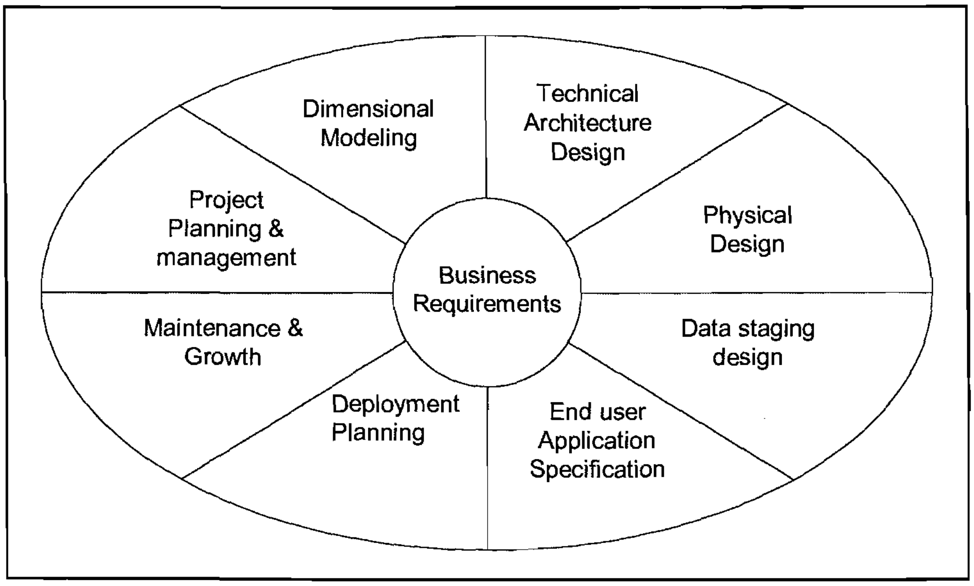


Figure 5-4 Business requirements impact every aspect of the data warehouse project (Kimball *et al.*, 1998:96)

This phase entails the foundation for the three parallel tracks focusing on technology, data and end user applications as depicted in Figure 5-3 (Kimball *et al.*, 1998:34).

5.3.1.3. Data Track: Dimensional Modelling

Once the business requirements are firmly established, the next step is to design the dimensional modelling.

Dimensional modelling is defined as "a logical design technique that seeks to present the data in a standard framework that is intuitive and allows for high performance access. It is inherently dimensional and adheres to a discipline that uses relational model with some important restrictions" (Kimball *et al.*, 1998:144).

Kimball *et al.* (1998:27) strongly believes that dimensional modelling should be part of the presentation phase of the data warehouse, as compared to entity-relationship (E/R) modelling, it yields better predictable and understandable designs, which can be used and assimilated by users and which can be queried effectively. Unlike E/R modelling, dimensional modelling does not require the database to be restructured, or queries to be rewritten when new data is introduced into the data warehouse. Lastly, a dimensional data mart does not need to anticipate the user's queries and is very resilient to changes in user analysis patterns. The explanation of dimension modelling is aided by Figure 5-5.

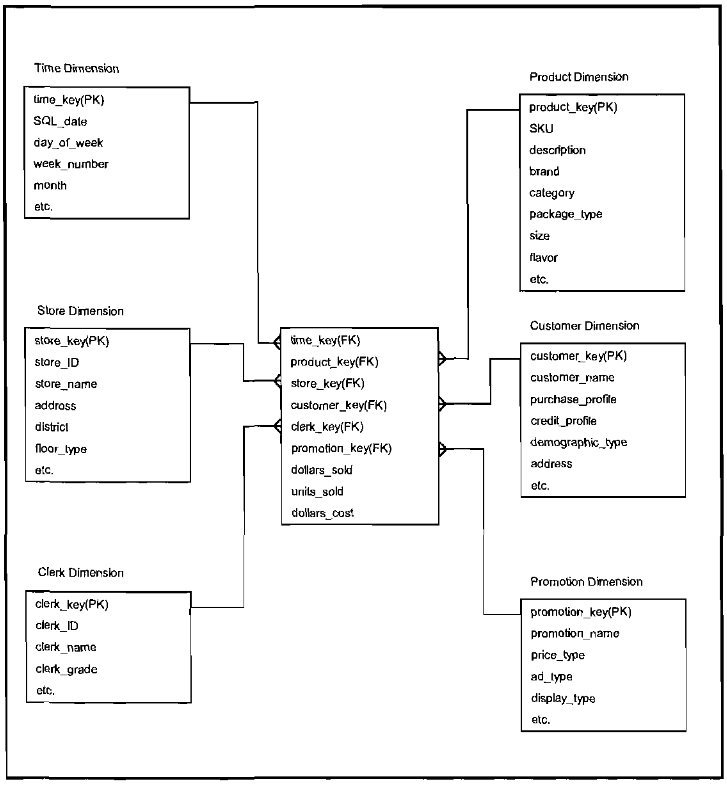


Figure 5-5 Example of a fact table (Kimball *et al.*, 1998:145)

Kimball *et al.* (1998:144) explains that this model consists of one table with a multipart key, called the fact table, and a set of similar tables, called dimensional tables. A dimensional table contains a single part primary key that corresponds to one of the components of the multipart key in the fact table. The fact table also contains one or more numerical facts. The authors (Kimball *et al.*, 1998:146) explain that the key to understanding the relationship between dimensional modelling and E/R modelling, is to take a single E/R diagram and break it down into multiple fact table diagrams. One then needs to select the many-to-may relationships in the E/R model containing numeric and additive non-key facts, and designate them as fact tables. Lastly, all the remaining tables need to be de-normalised into flat tables with single-part keys that connect directly to the fact tables. These tables become the dimension tables.

The master dimensional model of a data warehouse for a large enterprise will consists of 10 to 25 similar star-join schemas (Kimball *et al.*, 1998:146).

Dimensional modelling has five major strengths (Kimball *et al.*, 1998:147):

- A dimensional model has a predictable, standard framework, thus providing strong assumptions about the model and making interfaces more understandable.
- The predictable framework of the star join schema withstands unexpected changes in user behaviour
- Extensibility to accommodate unexpected new data elements and new design decisions. In summary, the following can be added:
  - New unanticipated facts that are consistent with the fundamental grain of the existing fact table.
  - New dimensions, for which a default value of that dimension is defined for the existing fact tables.
  - New unanticipated dimensional attributes.

    - Lastly, existing dimensional records taken down to a lower level of granularity from a certain point in time forward.
- A standard of approaches available for handling common modelling situations. These situations include:
    - Slowly changing dimensions – "constant" dimension that evolves asynchronously. Dimensional modelling provides specific techniques for handling slowly changing dimensions, depending on the business environment.
    - Heterogeneous products – for example, a bank needs to track a number of attributes and facts, but simultaneously needs to describe and measure the individual lines of business in highly idiosyncratic ways using incompatible facts.
    - Pay-in-advance databases – transactions of a business are not revenue, but the business needs to look at both the individual transactions, as well as reports on revenue on a regular basis.
    - Event-handling databases – the fact table is "factless".
- Growing strength of administrative utilities and software processes that manage the use of aggregates.

When creating a data warehouse, Kimball *et al.* (1998:156) advises that a set of standards must be formed. This set of standards is handled by the so-called Data Warehouse Bus Architecture, a master suite of conformed dimensions and standardised definitions of facts.

The authors recommend that the following should be kept in mind when creating conformed dimensions (Kimball *et al.*, 1998:156):

- Conformed dimension – dimension that produces the same meaning with every possible fact table to which it can be joined.

- Data warehouse design team – the responsible team to establish, publish, maintain, and enforce the conformed dimensions.
- Strict adherence to conformed dimensions – needed for the data warehouse to function as an integrated whole.

Conformed dimensions allow the following (Kimball *et al.*, 1998:157):

- A single dimension table to be used against multiple fact tables.
- User interfaces and data content to be consistent whenever a dimension is used.
- A consistent interpretation of attributes and therefore the ability to do rollups across data marts.

The Data Warehouse Bus Architecture matrix (Figure 5-6) is a tool to decide which dimensional model to build. The matrix requires the naming of all the data marts that can possibly be built, as well as the dimensions implied by those data marts (Kimball *et al.*, 1998:271).

	Time	Customer	Service	Rate Category	Local Service Provider	Calling Party	Called Party	Long-Distance Provider	Internal Organization	Employee	Location	Equipment Type	Supplier	Item Supplied	Weather	Account Status
Customer Billing	√	√	√	√	√			√			√					√
Service Orders	√	√	√		√			√	√	√	√	√			√	√
Trouble Reports	√	√	√		√	√		√	√	√	√	√	√	√	√	√
Yellow Page Ads	√	√		√		√			√	√	√					√
Customer Inquiries	√	√	√	√	√	√		√	√	√	√				√	√
Promotions & Comm'n	√	√	√	√	√	√		√	√	√	√	√	√	√		√
Billing Call Detail	√	√	√	√	√	√	√	√	√		√	√	√	√	√	√
Network Call Detail	√	√	√	√	√	√	√	√	√		√	√	√	√	√	√
Customer Inventory	√	√	√	√	√			√	√		√	√	√	√		√
Network Inventory	√		√						√	√	√	√	√	√		
Real Estate	√								√	√	√	√				
Labour & Payroll	√								√	√	√					
Computer Charges	√	√	√		√			√	√	√	√	√	√	√		
Purchase Orders	√								√	√	√	√	√	√		
Supplier Deliveries	√								√	√	√	√	√	√		
Combined Field Ops.	√	√	√	√	√	√		√	√	√	√	√	√	√	√	√
Customer Reln. Mgmnt	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Customer Profit	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

Figure 5-6 The Data Warehouse Bus Architecture matrix for a telephone company (Kimball *et al.*, 1998:271)

The rows of the above Figure 5-6 are the data marts and the columns the dimensions. The mark on the intersections indicates where a dimension exists for a data mart.

Advantages of the matrix are the following (Kimball *et al.*, 1998:272):

- It forces the question whether each candidate dimension might in some way be linked to a given data mart.
- It determines how important a dimension is by looking down the column.

Once all the potential data marts and dimensions are identified, the four-step method can be applied. The discussion on the four-step method for designing an individual fact table is based on Kimball *et al.* (1998:273).

Step 1: Choose the Data Mart

The data mart is selected. The following is recommended:

- The data warehouse designer should implement only single-source data marts first. This reduces the number of lengthy extract development tasks.
- Implement data marts in the context of a set of conformed dimensions to allow the plug in of the data marts into the data warehouse bus.

Step 2: Declare the Grain

The grain specifies the level of detail. It is advisable to be very precise when defining the fact table grain. The grain should be as low as possible to accommodate a more robust design. There are many advantages in choosing a low-level grain, such as individual transactions, individual day snapshots or individual document line items.

Step 3: Choose the Dimensions

The grain itself will often determine the primary or minimal set of dimensions needed. At this stage, the designer will examine all the data resources available and preferentially attach the single-valued descriptors as dimensions.

Step 4: Choose the facts

Add as many facts as possible within the context of the declared grain. The grain of the fact table allows the individual facts to be chosen and clarifies the scope of the facts. The facts should always be specific to the grain of the fact table.

5.3.1.4. Data Track: Physical Design

This phase focuses on the physical database design. The emphasis is on structures supporting the logical database design. Primary elements of this process will include defining, naming standards and setting up the database environment. Preliminary indexing and partitioning strategies are also determined (Kimball *et al.*, 1998:543).

A proper set of aggregated records that coexists with the primary base records, improves query performance. Four design goals should be kept in mind when aggregating a data mart (Kimball *et al.*, 1998:556):

- Aggregates must be stored in their own fact tables. The aggregated table should be separate from the base atomic data. Each distinct aggregation level must occupy its own unique fact table.
- The dimension tables attached to the aggregate fact tables must be shrunken versions of the dimension tables associated with the base fact table.
- The base atomic fact table and its related aggregated fact tables must be associated with one another. This is to allow the aggregate navigator to know which tables are related to one another.
- All created queries by any end user data access tool or application must be forced to refer exclusively to the base fact table and its full-size dimension tables.

To summarise the above discussion on completing the physical design, Kimball *et al.* (1998:571) recommends the following steps:

- Developing standards – this includes database-naming standards, using synonyms for all tables accessed by users, and develop standards for physical file locations.
- Developing the physical data model.
- Developing the initial index plan.
- Designing and building the database instance.
- Developing the physical storage structure.

- Implementing usage monitoring.

5.3.1.5. Data Track: Data Staging Design and Development

The data staging design and development phase of the business dimensional lifecycle represents the bulk of the data warehouse project (Kimball *et al.*, 1998:609).

A ten-step plan is followed to accomplish the data staging design and development. The discussion on the ten-step plan is based on Kimball *et al.* (1998:612-652).

The plan is divided into three sections:

- The plan
- Dimension loads
- Fact tables and automation.

The plan involves the following activities:

- Create a high-level, one-page schematic plan of the source-to-target flow. This schematic plan should be very simple. It should highlight what the developer knows about the source-to-target flow, where the data originates from, and annotate the major challenges that the developer may face. Figure 5-7 illustrates a typical example of this plan.

  The three steps in data staging, namely extract, transform and load are highlighted in this high level plan by illustrating the sources, transformations and targets.

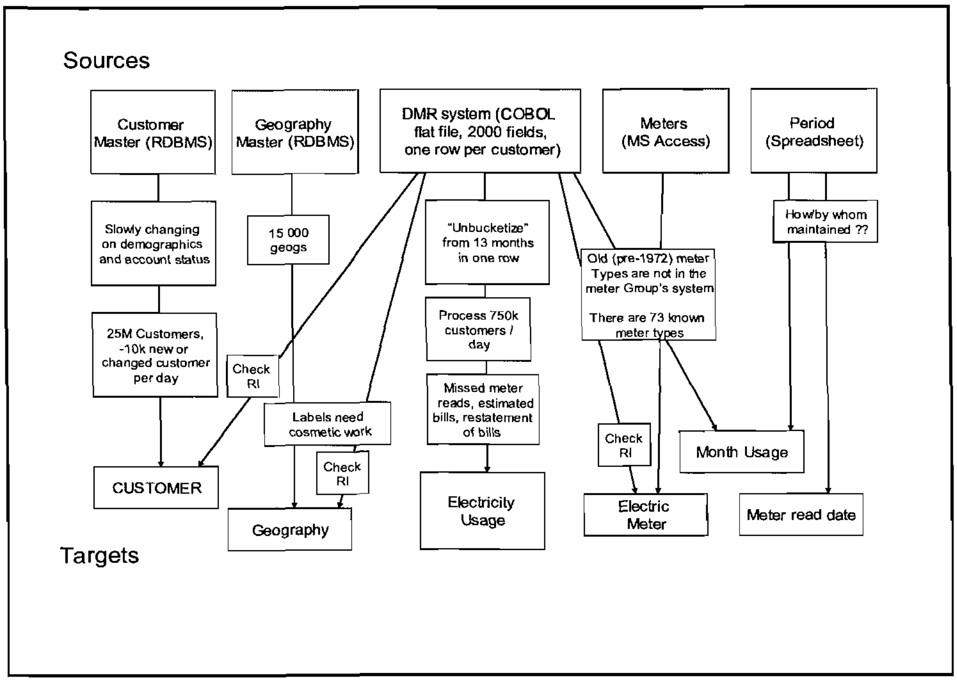

Sources
Customer Master (RDBMS)
Geography Master (RDBMS)
DMR system (COBOL flat file, 2000 fields, one row per customer)
Meters (MS Access)
Period (Spreadsheet)
Slowly changing on demographics and account status
15 000 geogs
"Unbucketize" from 13 months in one row
How/by whom maintained ??
Old (pre-1972) meter Types are not in the meter Group's system
There are 73 known meter types
25M Customers, -10k new or changed customer per day
Check RI
Process 750k customers / day
Missed meter reads, estimated bills, restatement of bills
Labels need cosmetic work
Check RI
Check RI
Month Usage
CUSTOMER
Geography
Electricity Usage
Electric Meter
Meter read date
Targets

Figure 5-7 Basic high-level data staging plan schematic (Kimball *et al.*, 1998:613)

- This is followed by testing, choosing and implementing a data staging tool. The decision of which tool to use, is based on the nature of the environment, cost and functionality.
- Create a detailed plan. A drill down by target table (changing the view of the data to a greater level of detail) and a graphical sketch detailing complex data restructures or transformations. The plan graphically illustrates the surrogate-key generation process which includes developing preliminary job sequencing.

Dimension loads involve the following activities.

- Build and test a static dimension table load. The primary goal of this step is to work out the infrastructure links, including connectivity, file transfer, and security problems.
- Build and test the slowly changing process for one dimension
- Build and test the remaining dimensions.

Fact tables and automation involves the following activities.

- Build and test the historical fact table loads (on base tables only) including surrogate-key lookup and substitution.
- Build and test the incremental load process and aggregate table loads.
- Design, build and test the staging automation.

5.3.1.6. Technology Track: Technical Architecture Design

An architectural plan is a technical translation of the business requirements. The essence of this phase is to identify the capabilities most important to the organisation. This is an iterative process, and as one moves forward, relevant information is uncovered and applied.

The value of architecture entails the following (Kimball *et al.*, 1998:318):

- Communication – provides a platform to communicate the project to management.
- Planning – provides a cross check for the project plan.
- Flexibility and maintenance – anticipate many possible issues and provide mechanisms for the possible issues.
- Learning – plays an important role as documentation for the system.
- Productivity and re-use – the architecture takes advantage of tools and metadata as the primary enablers of productivity and re-use.

The business requirements serve as the primary guide to what should be in the architecture and which parts should be prioritised. Once these are defined, the high-level technical architecture can be modelled (Kimball *et al.*, 1998:328). Figure 5-8 illustrates a high-level technical architecture.

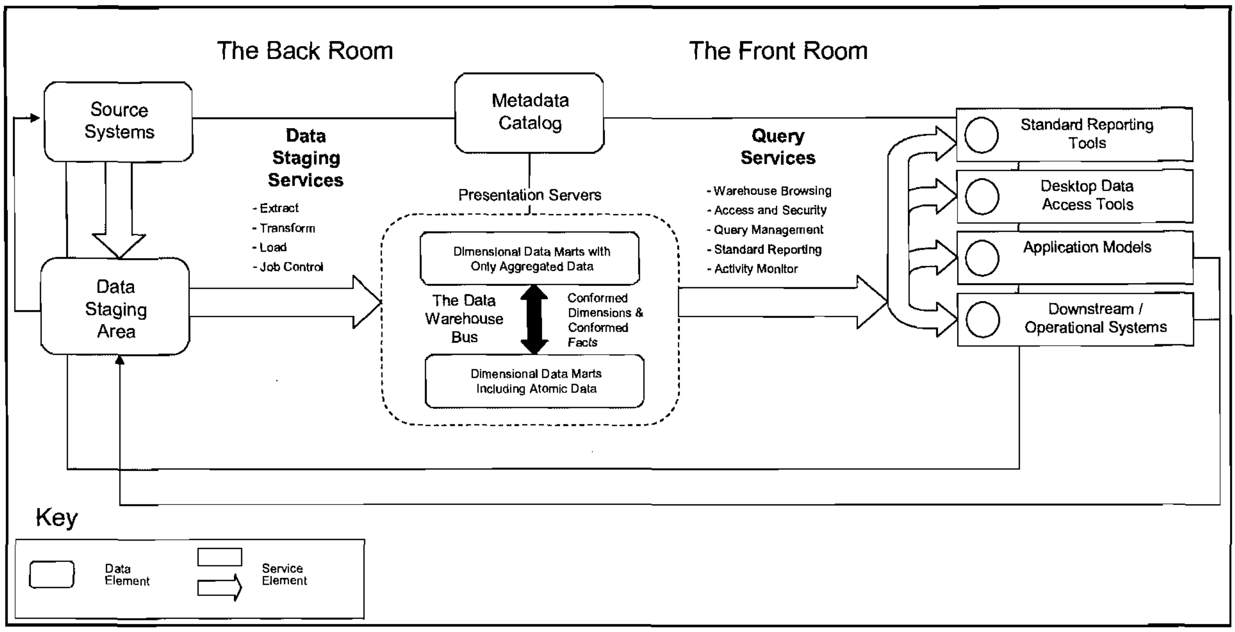


Figure 5-8 High-level technical architecture model (Kimball *et al.*, 1998:329)

The model provides a logical separation between the internal working of the warehouse, namely the back room and the front room. The back room is known for the area where the process of data acquisition is executed.

The backroom consists of the following (Kimball *et al.*, 1998:336):

- Source systems – these are typical transaction systems within the business, which can range from client/server ERP systems, reporting systems to operation data store.
- Data staging – described as the construction site of the warehouse, is the area where most of the data transformation takes place. Much of the value of the data warehouse is added at this stage. The advantages of using the data staging area are the following:
    - It provides a place for keeping emergency backup of the data.
    - The conformed dimensions are kept in flat files ready for export.
    - It is the source of most atomic transactional data.
- Presentation Servers – the target platform where direct querying is executed by end users. The Data Warehouse Bus in the presentation servers allows for

parallel development of business process data marts. The ability to integrate these data marts ensures the existence of conformed dimensions. The data marts found in the presentation servers do not show significant differences between the atomic data marts and the aggregated data marts, since all of the queries are still made in a dimensional format.

The second logical separation, the front room, is the part business interfaces with. The primary goal of the warehouse is to make information as accessible as possible; this is the function of the data access services layer. This layer reduces complexities between the data warehouse and the end users (Kimball *et al.*, 1998:373).

The access services provide for the following (Kimball *et al.*, 1998:378):

- Warehouse browsing – the data warehouse catalogue is used to support the users in their efforts to find and access information needed.
- Access and security services – facilitate the end user's connection to the data warehouse. These services rely on authentication from authentication services, a major design and management challenge.
- Activity monitoring services – involve capturing information about the usage of the data warehouse. This can be used for monitoring performance, user support, marketing and planning.
- Query management services – a set of capabilities that manages the exchange between query formulation, execution and results returned. These services are metadata driven, and services that are managed, are:
    - Content simplification
    - Query reformulation
    - Query retargeting and multi-pass SQL
    - Aggregate awareness
    - Data awareness
    - Query governing

- Standard reporting services – provide the ability to create a production style, fixed format report that has limited user interaction, a broad audience and regular execution schedules. It is described as a standard report of some sort.

Kimball *et al.* (1998:409) regards the front room development as vital, as this is the part of the data warehouse the business users interface with.

Figure 5-9 depicts the theory behind the architecture and illustrates the process flow of the architectural design. The high level model is entailed in the architecture process flow.

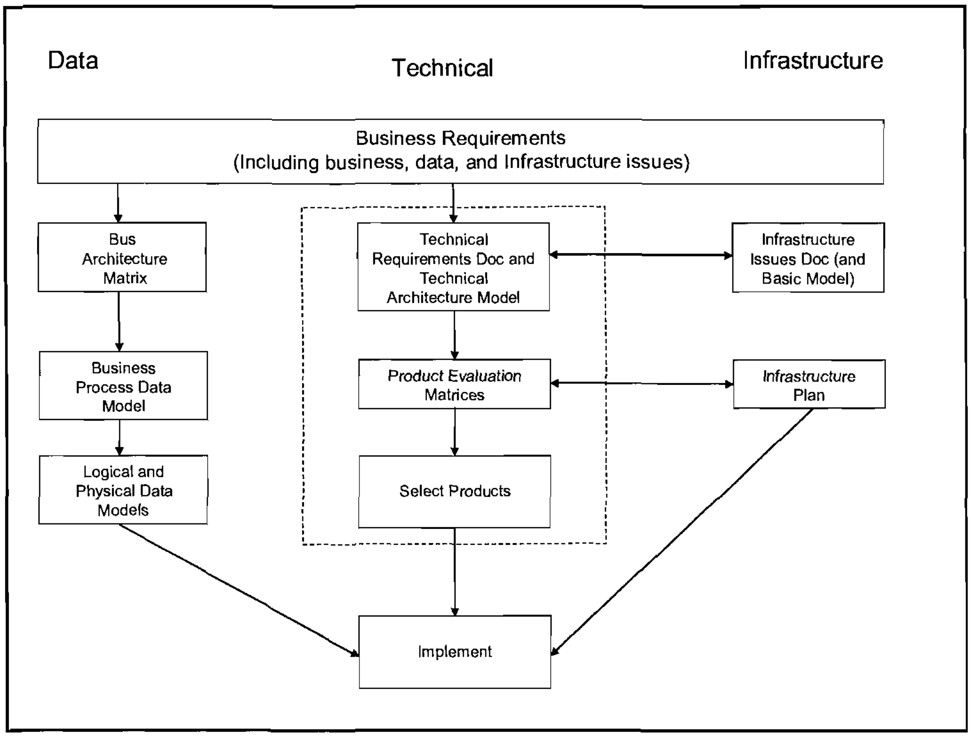

Data
Technical
Infrastructure
Business Requirements
(Including business, data, and Infrastructure issues)
Bus Architecture Matrix
Technical Requirements Doc and Technical Architecture Model
Infrastructure Issues Doc (and Basic Model)
Business Process Data Model
Product Evaluation Matrices
Infrastructure Plan
Logical and Physical Data Models
Select Products
Implement

Figure 5-9 Architecture development process flow chart (Kimball *et al.*, 1998:503)

The layer enclosed in the dashed-line box is the technical architecture; the process that needs to be followed in creating the technical architecture. The major deliverables in this process are the technical architecture plan and the infrastructure plan (Kimball *et al.*, 1998:504).

The process of creating the technical architecture should be executed once the requirements are clearly defined. This process includes (Kimball *et al.*, 1998:505):

- Form an architecture task force.
- Gather architecture-related requirements.
- Create a draft architecture requirements document.
- Create a technical architecture model.
- Determine the architecture implementation phases and deliverables for each phase.
- Create a technical architecture plan document.

5.3.1.7. Technology Track: Product Selection and Installation

During the selection of products, four major purchase areas for a typical warehouse must be considered (Kimball *et al.*, 1998:515):

- Hardware platforms
- DBMS platform
- Data staging tool
- Data access tools

To evaluate a product, the following evaluation process techniques are recommended (Kimball *et al.*, 1998:516):

- Production evaluation matrix – this contains certain product criteria that are evaluated to give an indication of which product is best suited to a certain situation.
- Market research

Once the above evaluation is completed, the selection can be narrowed down to no more than five products. The selected product can be installed and tested after an agreement has been reached. Thorough testing is required to ensure end-to-end integration of the data warehouse.

5.3.1.8. Application Track: End User Application Specification

The end user application fills a critical gap in meeting the data access needs of the organisation (Kimball *et al.*, 1998:666).

Four steps are identified to aid the specification process (Kimball *et al.*, 1998:670):

- Determine initial template set – including report candidate identification, as well as consolidation and prioritising of the list.
- Develop the navigation strategy – to assist users in finding what they need quickly. Using the template, metadata can also be useful.
- Determine template standards – naming and placing of objects and receiving an output that is satisfactory to the organisation.
- Develop detailed specifications – two parts of specifications are identified, namely the definition and the lay-out.

5.3.1.9. Application Track: End User Application Development

After the end user application specifications have been identified, the development of these applications can commence.

The development phase involves configuring the metadata tool and constructing the specified reports. This is done by using a data access tool. Kimball *et al.* (1998:678) states that the development lifecycle of the application depends on the organisation and the data access tool used.

5.3.1.10. Deployment Planning

Deployment planning is defined as "the convergence of technology, data and applications on the business users' desks, along with the necessary education and user support structure" (Kimball *et al.*, 1998:691).

Extensive planning is required before the actual deployment can commence. The following steps should be followed when planning deployment of a data warehouse (Kimball *et al.*, 1998:692):

- Determine desktop installation readiness – Kimball *et al.*(1998) point out that technology residing on the user's desktop, is the last piece that needs to be in place prior to deployment.
- Develop the end user education strategy – business users' education must address three key aspects of the data warehouse, namely:
  - Data content
  - End user application
  - The data warehouse access tool
- Develop an end user strategy – determining a support organisation structure, anticipation of data reconciliation support and end user application support, establishing support communication and feedback and lastly, providing support documentation.
- Develop the deployment release framework – including an alpha release, beta release and a production release framework.

5.3.1.11. Maintenance and Growth

Maintenance and growth are entered into, as soon as the data warehouse is operational. The following must be done to maintain the data warehouse (Kimball *et al.*, 1998:719):

- Manage the existing data warehouse environment – this focuses on the business users using the data warehouse and includes continued support and education.
- Manage the data warehouse operations – this includes managing the technical infrastructure, tuning the database performance and maintaining data and metadata management processes.

- Measure and market the data warehouse success – It is important to measure the performance of the data warehouse against the agreed success- and satisfaction criteria. In order to achieve this, one must do the following:
  - Monitor success and service metrics.
  - Capture the decisions made from using the data warehouse.
  - Proactively market the data warehouse.
- Communication – The maintenance plan should include an extensive communication strategy. This strategy should include the business sponsors and drivers, the business users, the general business community, the IS management and the data warehouse team.

5.3.1.12. Maintain and grow the data warehouse

A data warehouse is bound to evolve and grow; it is stated that "this is a sign of success, not failure" (Kimball *et al.*, 1998:727). To facilitate the ongoing development of the data warehouse, several strategies can be followed (Kimball *et al.*, 1998:728):

- Establishing a data warehouse steering committee.
- Prioritising growth and evolution opportunities.
- Managing iterative growth and evolution by using the business life cycle approach.

5.3.1.13. Project Management

Kimball *et al.* (1998:77) discusses the following techniques for keeping the data warehouse project on track.

- Conduct the project team kick-off meeting – The purpose of this "is to get the entire project team on the same page in terms of where the project currently stands and where it hopes to go" (Kimball *et al.*, 1998:78).
- Monitor project status – It should be monitored on a regular basis, by means of the following:
  - Project status meetings
  - Project status reports

- Maintain the project plan and project documentation – The integrated project plan mentioned earlier, should be updated weekly to accurately reflect progress and should be shared with the core project team.
- Manage the scope – There will be changes to the data warehouse project. They originate from two sources, namely previously unidentified issues and additional user requests.

The following techniques are recommended to track issues and change requests.

- Track issues – It is critical to ensure that nothing slips between the "cracks" and that everyone's concerns have been heard, also that the rationale used to resolve issues has been captured for future reference. Two classes of issues exist, firstly, those known to have an overall impact on the project and secondly, those known to be task-oriented. It is recommended that a log be kept to capture all issues.
- Control changes – "Formal acknowledgement of project changes is critical to overall project success. Any issues resolution that impacts the project schedule, budget or scope should be considered a change. If a change does not affect the schedule or budget, it is often not documented. However, these changes may still impact the overall scope. By formally documenting and communicating the change, users' expectations will be readjusted as well." (Kimball *et al.*, 1998:86)
- Document Enhancement requests – log/capture requests for future action, rather than expanding the current project scope.
- Develop communication plan to manage expectations – Establish a successful and robust communications plan ("game plan") to address the needs of different audiences. The overall plan should outline general message, content, format, and frequency of communication for each group of constitutes. Developing a communications plan, forces the project manager to fully consider the organisation's requirements. The following parties should engage in the communications plan:

- Project team
- Sponsor and drivers
- Business user community
- Other interested parties – these include executive management, IS organisation and the organisation at large.

5.3.2. Data driven methodology

Inmon (1996) follows the so-called data driven methodology to build data warehouses and defines a data warehouse as “a subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management’s decisions.”

To better understand this definition, one needs to investigate the key words. These are explained as follows: (Inmon, 1996:33)

- Subject-Orientation – A data warehouse is oriented around the major subjects of the organisation. It is organised around subjects such as customer, vendor, product and activity. The grouping around subject areas affects the design and implementation of the data found in the data warehouse.
- Integration – Data found in the data warehouse environment is integrated. This is the essence of the data warehouse environment. Integration can be implemented in different ways, such as consistent naming conventions, consistent measurement of variables, consistent encoding structures, consistent physical attributes of data and so forth.
- Non-volatile – Inserts, deletes and changes are done regularly to the operational environment on a record-by-record basis. Data manipulation occurring in the data warehouse is much simpler. There are only two kinds of operations that occur in the data warehouse, i.e. the initial loading of data and the access of data. There is no update of data.
- Time variance – All data in the data warehouse is accurate as of some moment in time. This basic characteristic of data in the warehouse is very different from data found in the operational environment. In the operational environment, data

is accurate as of the moment of access. In the data warehouse, data is accurate as of some moment in time. Data found in the data warehouse is said to be "time-variant".

Inmon (1996:291) makes it clear that the Decision Support System (DSS) does not take on a normal operational development life cycle, starting with requirements and ending with the code, as it starts with data and ends with requirements.

Inmon presents the methodology in three parts;

- The first part is aimed at operational systems and processing – It includes interviews to produce a "soft core" of what the production system does, as well as determining the opinion of middle management and data gathering, detailed to "fill in the gap" in the requirements process. Furthermore, Joint Application Design (JAD) sessions to execute group brainstorming for spontaneous flow of ideas and strategic business planning to manifest the system in business. (Inmon, 1996:317)
- The second part is aimed at DSS systems and processing – This is the methodology used to create the data warehouse.
- The final part is aimed at the heuristic component of the development process – It focuses on the usage of the data warehouse for the purpose of analysis. This main difference in this phase is that the development process always starts with the data from the data warehouse. Secondly, the requirements are not known at the beginning of the development process and thirdly, the processing is done in a very iterative heuristic fashion. (Inmon, 1996:344)

This research will be focussing on the second part of Inmon's (1996) data driven methodology, namely the development of DSS systems, which entails the following:

- Data model analysis
- Breadbox analysis

- Technical analysis
- Subject area analysis
- Data warehouse design
- Source system analysis
- Specifications
- Programming
- Population

5.3.2.1. The data model analysis

The data model analysis is the first activity in the development of a DSS system. According to Inmon (1996:335), the data model takes on different levels of modelling (Figure 5-10), called corporate modelling, operational data modelling and data warehouse modelling.

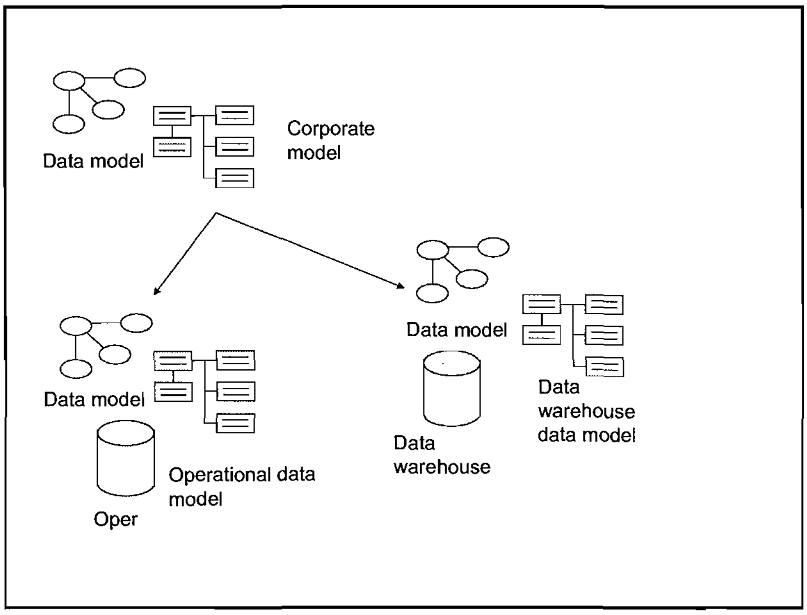


Figure 5-10 The relationship between the corporate data model and the operational model and data warehouse model (Inmon, 1996:83)

The corporate model only contains primitive data and is constructed with no distinction between existing operational systems and the data warehouse. Performance factors are added to the corporate model to transport it to the operational data model. The last model is the data warehouse model, which contains a fair number of changes. It is also developed in an iterative fashion.

The following changes are applied to the data warehouse model:

- Data used purely in the operational system, is removed.
- The key structures of the corporate data model are enhanced with an element of time.
- Derived data is added to the corporate data model where the derived data is publicly used and calculated once only and not repeatedly.
- Data relationships in the operational environment are turned into artefacts in the data warehouse.
- A stability analysis is performed to the data warehouse data model.

A stability analysis is a technique to identify attributes with similar characteristics. Figure 5-11 is an example of a stability analysis. It illustrates the following attributes:

- Description
- Lead time
- Acceptable credit rate
- Shipping manifest

These attributes do not change frequently, thus they are grouped together. The next set of attributes (Primary substitute, Safety stock, Primary supplier, Expediter) changes more frequently than the first group and is therefore grouped together. The last group (Quantity on hand, Order unit, Last order date, Last delivery and Order amount) changes most frequently and is grouped together. The Part-id attribute serves as a key and is included in all the groups.

A data model has three levels (Inmon, 1996:85):

- High level modelling
- Mid level modelling
- Low level modelling

High level modelling is called the Entity Relationship Diagram (ERD). It represents the entity relationship level. Entities shown in the ERD are at the highest level of abstraction. The choice of what entities belong to the scope of the model and what entities do not is determined by what Inmon calls the "scope of integration". The so-called scope of integration defines the boundaries of the data model and needs to be defined before the modelling process commences. The scope of integration should be agreed upon by the modeller, management and the ultimate user of the system.

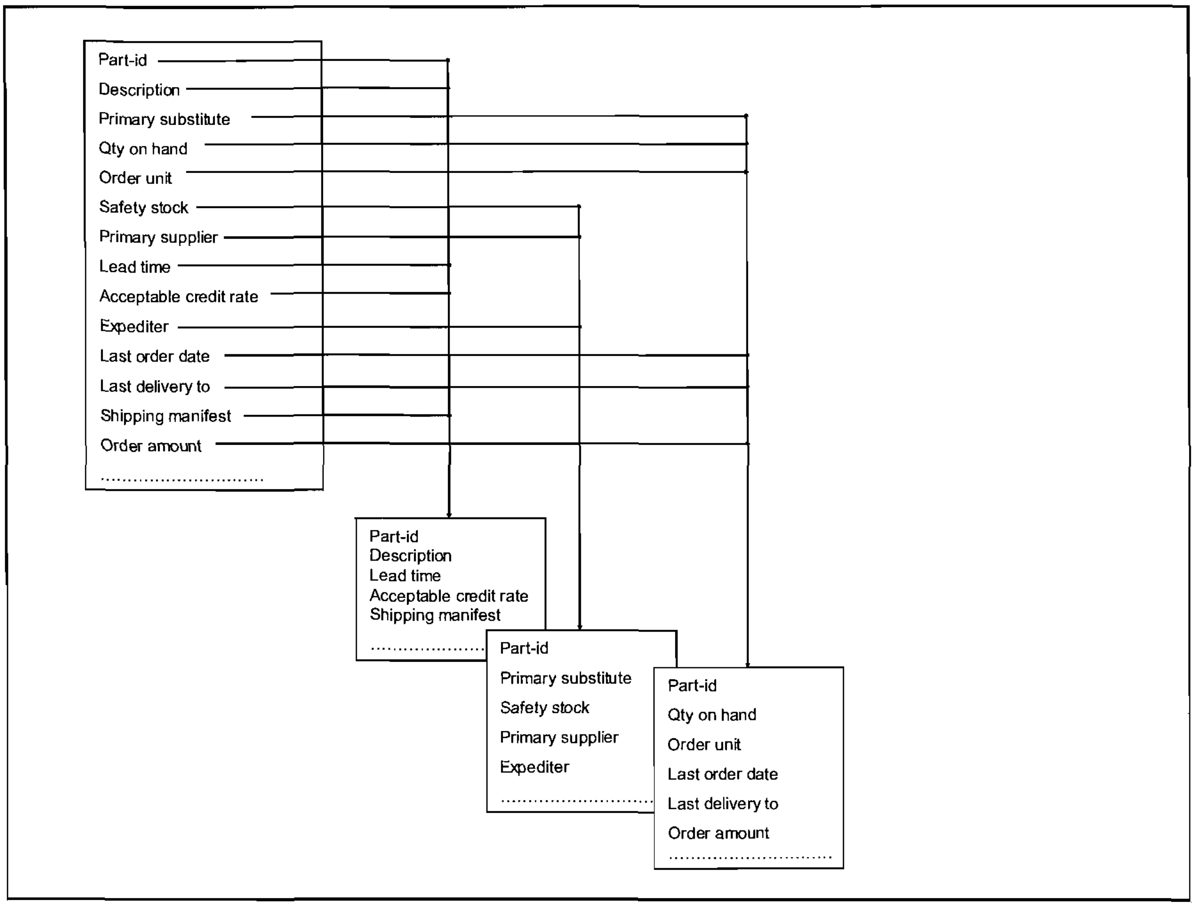


Figure 5-11 Stability Analysis performed on a table (Inmon, 1996:84)

Mid level modelling is called the Data Item Set (DIS). This is the next model after the high level model. For each subject area or entity identified in the high level model, a mid level model is created. Figure 5-12 is an example of a data item set.

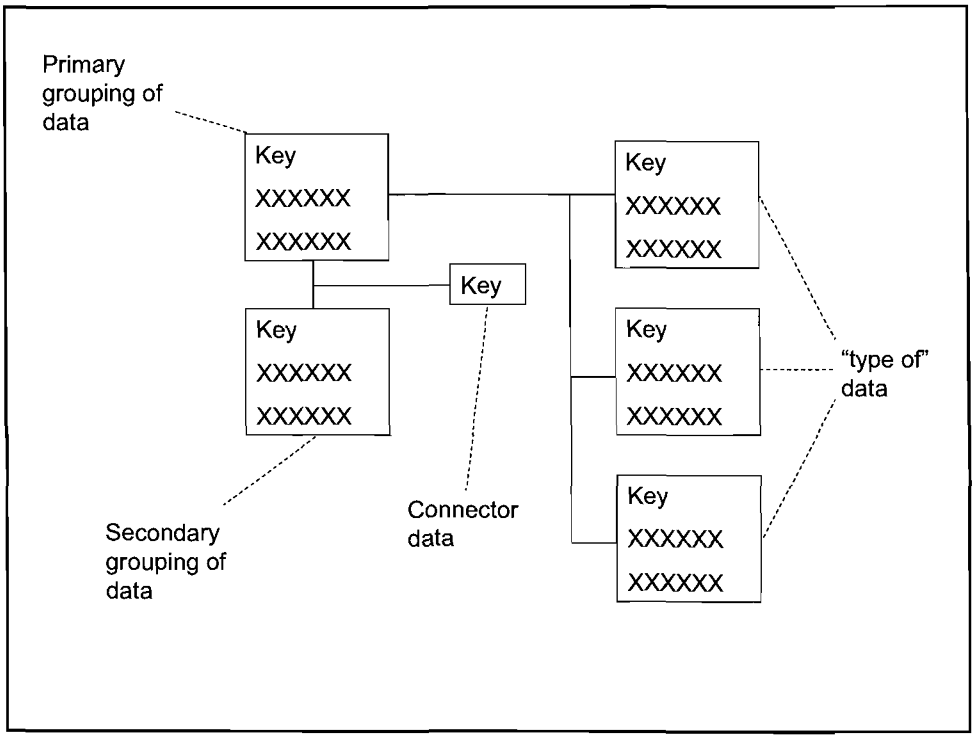


Figure 5-12 Example of a data item set (Inmon, 1996:89)

The primary grouping exists only once for each major subject area and holds attributes that exist only once for each subject area. The primary grouping contains attributes and keys. The secondary grouping of data holds data attributes that can exist multiple times for each major subject area. The secondary grouping is indicated by a line emanating downward from the primary grouping of data. The connector relates data from one grouping to another. The "type of" data is indicated by a line leading to the right of a data grouping. The grouping of data to the left is the super type of data. The grouping of data to the right is the sub-type of data.

The lowest level is the low level model. This is called the physical model. This model is created from the mid level data model. Inmon (1996:96) indicates that this model may look like relational tables. Figure 5-13 is an example of a physical model.

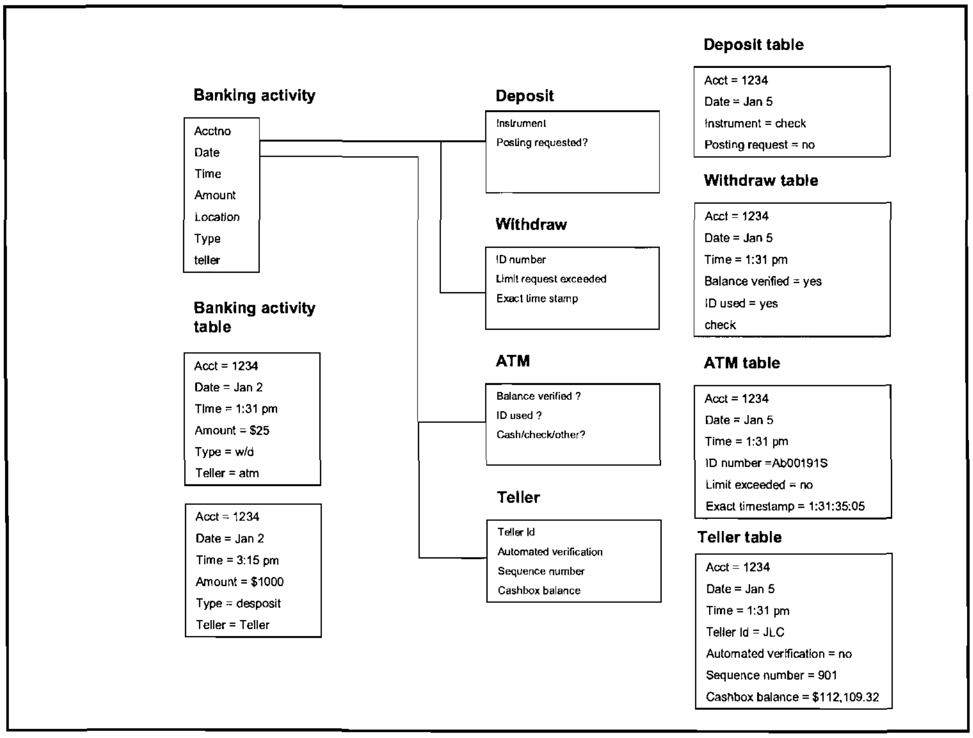


Figure 5-13 Example of a physical model (Inmon, 1996:93)

Inmon (1996:335) states that a data model (data warehouse) is successful when it satisfies the following requirements:

- Major subject areas are identified
- Boundaries of the model are clearly defined
- Primitive data is separated from derived data
- The following are identified for each subject area:
  - Keys
  - Attributes
  - Groupings of attributes
  - Relationships among groupings of attributes
  - Multiple occurring data

- Type of data

The end result of the data model is a series of tables, each of which contains keys and attributes. To save input/output, tables can be normalised/de-normalised. Unfortunately, there is no strategy to follow, and as Inmon explains "it is answering this question that the physical database designer earns his or her reward" (Inmon, 1996:93).

5.3.2.2. Breadbox analysis

After the data analysis is completed, the next step is to carry out a breadbox analysis. A breadbox analysis is a sizing (in terms of gross estimates) of the DSS environment and the project, as well as how much data the data warehouse will hold (Inmon, 1996:336).The parameter for success is to estimate the amount of data (in terms of number of rows) on both the one-year and five-year horizons for the entire data-warehouse environment. Based on the results of the estimate, one decides whether different levels of granularity are needed. If the data warehouse needs to contain large amounts of data, multiple levels of granularity should be considered. If the data warehouse is small, multiple levels are not required.

5.3.2.3. Technical assessment

Inmon (1996:337) argues that the technical requirements for managing the data warehouse are different from the technical requirements for managing an operational database. Consideration is needed for managing data and processing it in the operational environment.

To be successful, the technical functionality of the data warehouse must be able to:

- Manage large amounts of data.
- Allow data to be accessed flexibly.
- Organise data according to a data model.
- Both receive and send data to a wide variety of technologies.
- Have data periodically loaded in masses.

- Access a set at a time, or a record at a time

5.3.2.4. Subject area analysis

During the subject area analysis, the subjects to be populated are selected. The subject area should be large enough to be meaningful and small enough to be implemented with ease (Inmon, 1996:339). The populations should start small and can follow larger subjects, or even sub-sets of subjects. The output from this phase should serve as a definition of what data is to be populated.

5.3.2.5. Data warehouse design

The core step in the development of the data warehouse is the data warehouse design. The data warehouse design should not be done in a heuristic manner, but should be done using the feedback loop (Inmon, 1996:73):

- Portion of the data is populated.
- The data is then used and scrutinised by the DSS analyst.
- Based on the feedback of the end user, the data is modified, or other data is added to the data warehouse

It is argued that the requirements of the data warehouse cannot be identified, thus requirements driven approaches will not help. For this reason, loop needs to be continued throughout the entire life of the data warehouse.

The data warehouse design is based on the data model. This model represents the needs of the organisation and is a technology in dependent view (Inmon, 1996:276). A few aspects of the data model need to be changed to turn the data model into a data warehouse design (Inmon, 1996:278):

- An element of time needs to be added to the key structure if it is not already present.
- All purely operational data needs to be eliminated.
- Referential integrity relationships need to be turned into artefacts.
- Derived data frequently used, is added to the design.

- The structure of the data needs to be altered when appropriate for:
  - Adding arrays of data.
  - Adding data redundantly.
  - Further separating data under the right conditions.
  - Merging tables when appropriate.
- Stability analysis of the data is done.

The data warehouse design will typically be organised around the subject areas identified.

Data in the data warehouse is stored in a multidimensional perspective (Inmon, 1996:140). Figure 5-14 illustrates a three dimensional view of data in the data warehouse; it illustrates that the entities (vendor, customer, order, shipment and product) do not contain equal amounts of records.

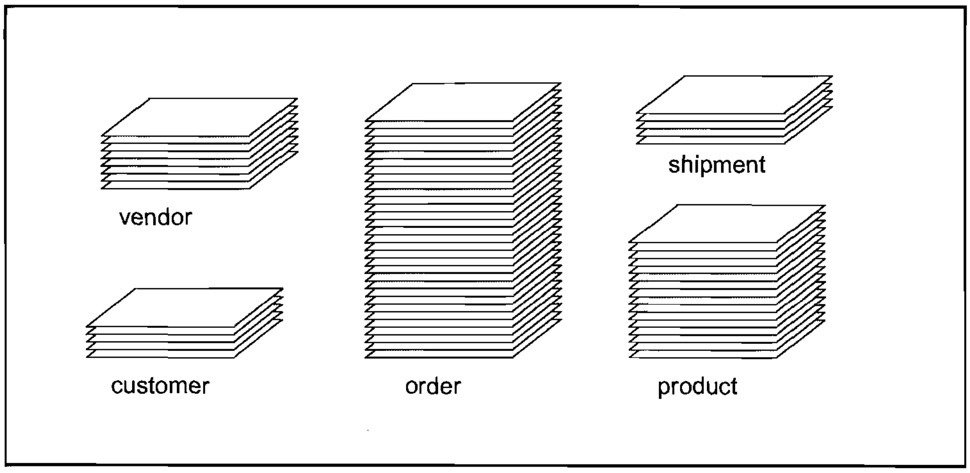


Figure 5-14 A three dimensional view of data in the data warehouse (Inmon, 1996:140)

One shortcoming of a data model is that it cannot represent data that should be multidimensional (Inmon, 1996:139).This shortcoming is resolved by using a technique called star joins (Inmon, 1996:140).The star joins-technique enables one to manage large amounts of data residing in an entity in the data warehouse. Figure 5-15 shows a simple star join in which one entity is populated.

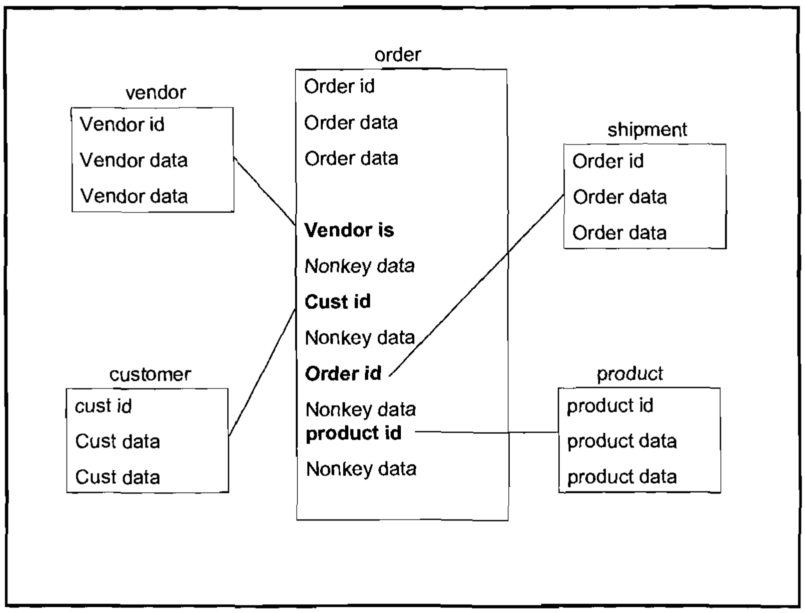


Figure 5-15 A star join of the order entity (Inmon, 1996:141)

In Figure 5-15, ORDER is at the centre of the star join. ORDER is the entity that will be heavily populated. Surrounding ORDER, are entities PART, DATE, SUPPLIER and SHIPMENT. Each of the surrounding entities will have only a modest number of occurrences of data. The centre of the star join, ORDER, is called a “fact table”. The surrounding tables are called “dimension tables”. The fact table contains unique identifying data for ORDER, as well as data unique to order itself. The fact table also contains pre-joined foreign key references to tables surrounding it, the dimension tables. Derived data frequently needed, is added to the design. This data will be added to the fact table. Inmon (1996:142) highlights that textual data is located in the dimension tables, whereas numeric data is located in the fact table.

The above step should produce a physical database design of the data warehouse. The entire data warehouse needs not to be designed in detail. It is entirely acceptable to initially design the major structures of the data warehouse, and then fill in the detail at a later point in time (Inmon, 1996:340).

5.3.2.6. Source system analysis

In this activity, one should define the system of record. The so-called system of record is defined as "nothing more than the identification of the "best" data the corporation has" (Inmon, 1996:276). The "best" source of data is determined by the following criteria in the existing systems environment (Inmon, 1996:276):

- What data is the most complete?
- What data is the most timely?
- What data is the most accurate?
- What data is the closest to the source of entry into the existing systems environment?
- What data conforms closest to the structure of the data model in terms of keys, attributes, or groupings of data attributes together?

This is also the point where issues of integration are addressed. These issues include (Inmon, 1996:341):

- Key structure/key resolution for the data that passes from the operational environment to the DSS environment.
- Attributes:
    - How to choose from multiple sources.
    - How to handle situations where no sources are available to choose from.
    - How to transform selected data for the DSS environment.
    - What transformations are needed e.g. encoding/decoding, conversions, etc, or must be made as data is selected for transport to the DSS environment?
- How the time variance will be created from current data.
- How the DSS structure will be created from the operational structure.
- How operational relationships will show in the DSS environment.

The activity discussed, will provide a mapping of data from the operational environment to the DSS environment.

5.3.2.7. Specifications

Once the interface between the operational environment and the DSS environment has been outlined, the next step is to formalise it in terms of programming specifications.

Some of the major issues include the following: (Inmon, 1996:342)

- Which operational data should be scanned?
  - Is the operational data time stamped?
  - Is there a delta file?
  - Are there system logs/audit logs that can be used?
  - Can existing source code and data structure be changed to create a delta file?
  - Do before- and after image files have to be rubbed together?
- How should the output be stored once scanned?
  - Is the DSS data reallocated, preformatted?
  - Is data appended?
  - Is data replaced?
  - Are updates in the DSS environment made?

Inmon (1996:342) advises that the output from this step is the actual program specifications that will be used to bring data over from the operational environment to the data warehouse.

5.3.2.8. Programming

Once the source system analysis is completed and the actual program specifications are gathered, one can commence with the programming phase. According to Inmon, this includes activities such as the following (Inmon, 1996:343):

- Development of pseudo code
- Coding

- Compilation
- Walkthroughs
- Testing

5.3.2.9. Population

In population, the programs previously developed for the DSS are executed. Inmon (1996:281) recommends that the smaller subject area should be populated first, whilst the larger subject area should be partially populated. It is done because of a significant possibility that the requirements may change. The population of subject areas typically works in the feedback loop process. Figure 5-16 is an illustration of the feedback loop.

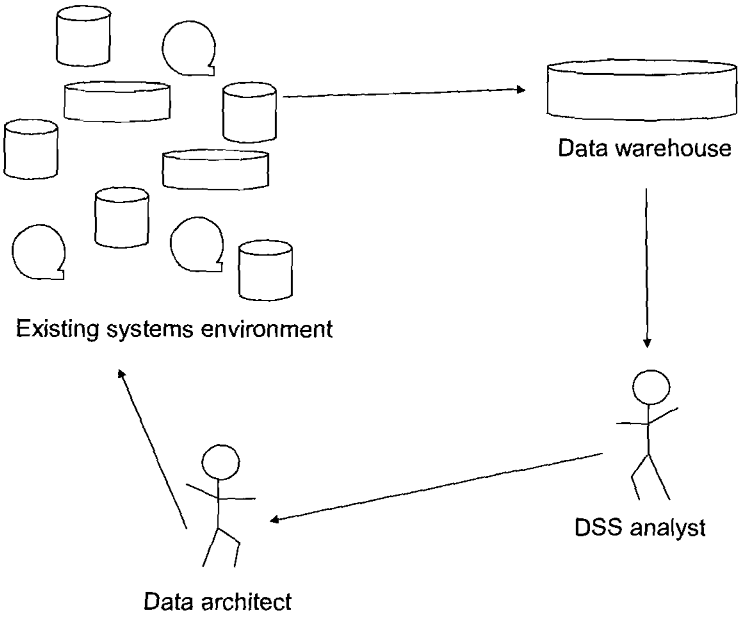


Figure 5-16 The feedback loop (Inmon, 1996:283)

The following issues are addressed in the feedback loop. (Inmon, 1996:283)

- "Frequency of population.
- Purging populated data.
- Aging populated data (i.e. running tally summary programs).
- Managing multiple levels of granularity.

- Refreshing living sample data (if living sample tables have been built)."

The result of this phase should provide a populated functional data warehouse.

5.4. Summary

This chapter covered the development approaches used in data warehouse development.

Although these approaches seem to be quite contrasting, Kimball *et al.* (1998:18) respond by stating that they do not believe there are two "contrasting" points of view about top-down and bottom-up data warehouses.

According to Kimball *et al.* (1998) the extreme top-down perspective is a completely centralised, tightly designed master database that must be completed before the parts are summarised and published as individual data marts.

The second approach mentioned by Kimball *et al.* (1998) *is that* the extreme bottom-up perspective is an enterprise data warehouse that can be assembled from disparate and unrelated data marts. The authors also state that neither of these approaches taken to the limit is feasible and that the only workable solution is to blend the two.

Chapter 6 - Data Warehouse and the Object Oriented Approach

6.1. Introduction

This chapter illustrates how to develop data warehouses by using (OO) concepts, tools and techniques where possible.

In Chapter 4, OO analysis as the first phase of OO development was discussed. Booch (1994:155) states that analysis is to model the world by discovering the classes and objects that form the vocabulary of the problem domain. In design, one invents the abstraction and mechanisms that provide the behaviour the model requires.

The data warehouse (DW) business dimensional lifecycle approach (Kimball *et al.*, 1998:33) starts with the business requirements definition in an attempt to gather the requirements needed for the development. The business dimensional lifecycle approach is a requirements-based methodology.

The DW data-driven methodology (Inmon, 1996:291) on the other hand starts with the data and does not follow a requirements-driven approach in its development methodology. The fact that the data-driven methodology is not requirements-driven does not necessarily mean that it is not compatible with OO development methodology.

6.2. The OO model

Chapter 4 covered the OOA, OOSP, RUP and OMT methodologies. It concluded with a comparison between the different methodologies and the commonalities found between them. Figure 6-1 is an illustration of the common activities found in each phase.

Phase		Activity
Requirements	Gathering Requirements	• Essential Use Case Diagram • Essential Use Case • User Interface Prototyping • Domain Modelling • Supplementary Documentation
	Validating Requirements	• Use Case Scenario Testing • User Interface Walkthrough • Requirements Review
Object Oriented Analysis		• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages
Object Oriented Design		• Class Modelling • State Chart Modelling • Collaboration Modelling • Component Modelling • Deployment Modelling • Relational Persistence Modelling • User Interface Design
Object Oriented Implementation		• Code development • Component packaging • Deploy packages
Object Oriented Testing		• Unit test components

Figure 6-1 Summary of requirements, analysis, design, implementation and testing.

This chapter describes the phases of the DW lifecycle following an OO approach. The OO approach is based on the phases and techniques illustrated in Figure 6-1.

6.3. Data warehouse (DW) development using the Business dimensional lifecycle approach phases and an OO approach.

The objective of this discussion is to describe how a data warehouse (DW) is built, should one use the approach of Kimball *et al.* (1998) and implement it in an OO fashion.

The Business Dimensional Lifecycle of Kimball *et al.* (1998, 33) is illustrated in Figure 6-2. The figure also shows the paragraph numbers of the following discussion.

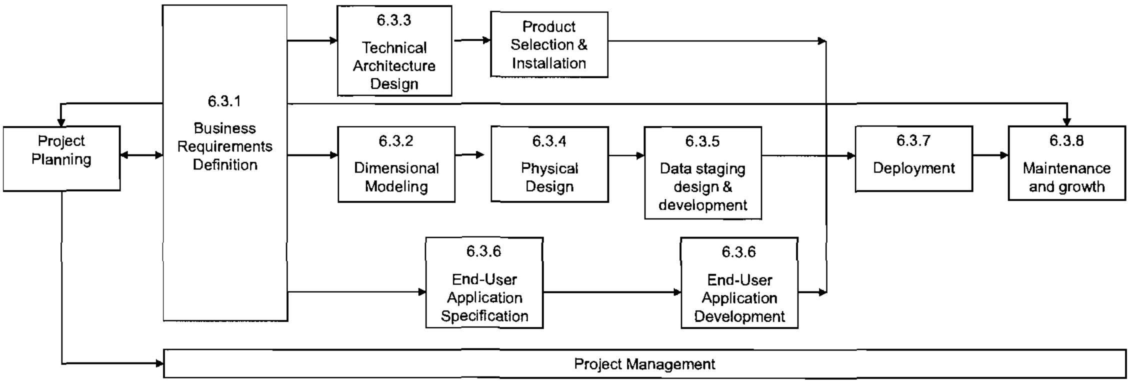

6.3.3
Technical Architecture Design
Product Selection & Installation
Project Planning
6.3.1
Business Requirements Definition
6.3.2
Dimensional Modeling
6.3.4
Physical Design
6.3.5
Data staging design & development
6.3.7
Deployment
6.3.8
Maintenance and growth
6.3.6
End-User Application Specification
6.3.6
End-User Application Development
Project Management

Figure 6-2 Business Dimensional Lifecycle diagram (Kimball *et al.*, 1998:33)

The discussion on project planning and project management given by Kimball *et al.* (1998) is generic and not focused on DW projects exclusively. This chapter focuses on DW-specific development techniques from an OO perspective, therefore project planning is omitted from this discussion.

The discussion of building a DW using the Kimball *et al.* (1998) approach and implementing it in an OO fashion follows the structure illustrated in Figure 6-2.

The first part of the discussion (section 6.3.1) concentrates on OO methods for acquiring the business requirements needed to build the DW. This is followed by a discussion on OO methods used to analyse and design the DW dimensional models (section 6.3.2) and the technical architecture (section 6.3.3) relevant to the defined business requirements. Following on, are discussions on the physical design of the DW (section 6.3.4) and the design, implementation and testing of the data staging area (section 6.3.5). The section is concluded by short discussions on the full development lifecycle of the end user applications (section 6.3.6), deployment of the DW (section 6.3.7), as well as its maintenance and growth DW (section 6.3.8).

6.3.1. Business Requirements Definition

The Business Requirements Definition determines:

- Data that should be available in the DW.
- How it should be organised.
- How often it should be refreshed (Kimball *et al.*, 1998:95).

In OO terms, gathering the business requirements starts with a requirements modelling team (Ambler, 2001:34). An interview team (Kimball *et al.*, 1998:98) is used to gather requirements for the DW. From a DW perspective, both these terms focus on generating business requirements information.

Requirements gathering techniques are:

- Interviewing
- Facilitated sessions
- Brainstorming

In conjunction with interviewing and brainstorming, techniques in the requirements model derived in chapter 4, can be applied. Figure 6-3 is an illustration of these techniques.

Phase		Activity
Requirements	Gathering Requirements	• Essential Use Case Diagram • Essential Use Case • User Interface Prototyping • Domain Modelling • Supplementary Documentation
	Validating Requirements	• Use Case Scenario Testing • User Interface Walkthrough • Requirements Review

Figure 6-3 Requirements Model in OOD

The requirements phase in object orient development (OOD) consists of two sub-phases, namely:

- Requirements gathering
- Requirements validation

The following discussion will explain each technique in the requirements model, as well as an application of the technique in terms of data warehouse development.

6.3.1.1. Essential use case diagram

An essential use case diagram is a technology-independent view of behavioural requirements and describes the details between the user and the system (Ambler, 2001:44). As discussed in chapter 4, its function is to illustrate the interaction between actors and concepts in the problem domain.

A DW is a system that allows the user to access the "bigger picture" of the business data. To make DW development achievable, limits must be set to what should be included in the "bigger picture", as well as to the inputs and outputs of components that make up the "bigger picture".

The essential use case diagram can be altered to illustrate which components of the "bigger picture" are represented in the data warehouse, and who will use what information. To avoid confusion between the essential use case diagram representing a specific business problem domain and the essential use case diagram used to represent the "bigger picture" of the business, they will be referred to as the business process use case diagram and the data warehouse use case diagram respectively. Figure 6-4 is an example of a data warehouse use case diagram.

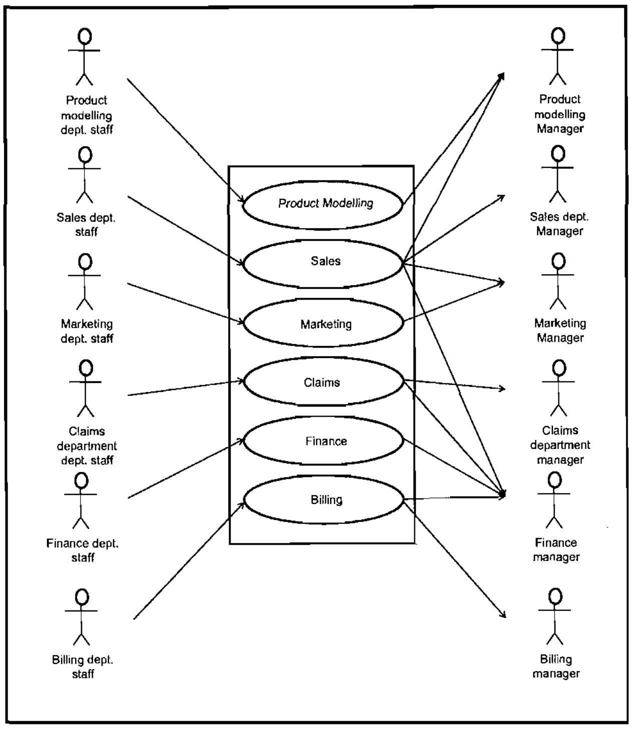


Figure 6-4 Data warehouse use case diagram

The ellipses in the data warehouse use case diagram represent the different departments within the organisation. An analysis is done on the departments in order to identify specific business processes that will be supported by the data warehouse. The actors on the left hand side of the diagram represent the parties responsible for generating the information. The information technology (IT) department is excluded as the responsibility of this department is to process information, not to generate it. In cases where the IT department is responsible for generating information, it will be included. The actors on the right hand side represent the parties who need to receive the information from the data warehouse. They will typically include the manager of the respective departments and in some cases the board of directors, depending on how the company is structured. The concept of the data warehouse use case diagram is to create a picture of the different business units (departments), indicating who is responsible for generating inputs for a business unit and who is using the output of a business unit (users). Table 6-1 is an example of the use cases generated from the data warehouse use case diagram.

DW Use Case Number	Department	Party Responsible for providing information	Users
DWUC01	Product modelling	Product modelling staff	Product modelling manager
DWUC02	Sales	Sales staff	Product modelling manager, Sales manager, Marketing manager and Finance manager.
DWUC03	Marketing	Marketing staff	Marketing manager
DWUC04	Claims	Claims staff	Claims manager, Finance manager
DWUC05	Finance	Finance staff	Finance manager
DWUC06	Billing	Billing staff	Finance manager and Billing manager

Table 6-1 Example of a list of data warehouse use cases generated from the data warehouse use case diagram.

Each of the DW use cases contains one or multiple business processes. The business essential use case diagram represents these business processes. Figure 6-5 illustrates an example of an essential business process use case diagram representing the process of quoting for insurance cover for an insurance company.

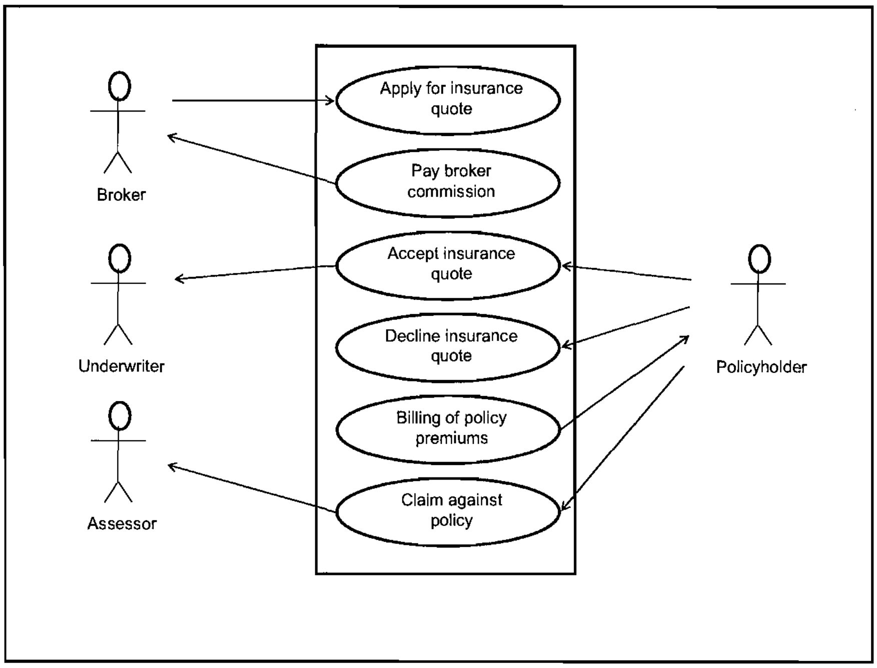


Figure 6-5 Essential business process use case diagram example for an insurance company

The link between the actor and the ellipse in the diagram represents an interaction. From this interaction one is able to create an essential use case. Essential use cases are created for every interaction in the diagram

Use case number	Description
UC01	Broker requests a quote for insurance policy for a prospective policy holder
UC02	Broker receives commission for a successful application
UC03	Prospective policyholder accepts insurance policy quote
UC04	Prospective policyholder declines insurance policy quote
UC05	Insurance company bills policyholder for monthly premiums
UC06	Policyholder claims against policy
UC08	Assessor assesses policy claim

Table 6-2 Example of a list of essential use cases generated from the use case diagram

Table 6-2 illustrates the use cases that can be generated from the essential use case diagram. The idea of the essential use case diagram is to make the requirements team aware of all possible use cases within the problem domain.

The essential use case diagram illustrated in Figure 6-5 represents the business from a production point of view. The purpose of this illustration is to identify the core business processes, as well as the actors involved in these processes.

6.3.1.2. Essential use case

The essential use case is also divided into a business essential use case containing the derived information from the business process use case diagram and a data warehouse essential use case derived from the data warehouse use case diagram.

Data warehouse essential use case

Data warehouse essential use case contains the name, description, inputs, outputs and the grain needed for this use case. Figure 6-66 is an example of a DW use case.

Name: DWUC 02 Sales

Description: Sales department responsible for sales of a product

Basic inputs

Sales staff should provide the following:

1. Sales of each product
2. Possible dates of sold product
3. Possible regions of sold product
4. Possible brokers responsible for sales in regions

Basic outputs

1. Regional sales figures

Possible grain(s)

1. The sale of a product.

Figure 6-6 Data warehouse use case example for sales department

The inputs and outputs are derived from the DW use case diagram. The possible grain heading is added to the DW essential use case to serve as an indication of the lowest level of information contributing to the core function of the department.

For example, the grain in Figure 6-66 is recorded as the sale of a product, thus one sale of one item equals the grain. If all of the items sold are added up, they will make up the total sales. Grain is a DW-specific addition to the standard use case definition.

Business essential use case

The business essential use case contains the name of the use case, the description of the use case, the precondition, post conditions and the course of action. Action between components in the use case is hereby explained. Figure 6-7 illustrates the business essential use case for a member requesting a quote for insurance purposes.

Name: UC 01 Quote for insurance.

Description: Broker requests a quote for insurance policy for a prospective policy holder

Preconditions (Business rules):

Policyholder must be older than 21 years
Policyholder must be female for maternity products
Policyholder can only have one life assurance product
Policyholder must be quoted via an accredited broker
...
...

Post conditions:

Member accepts the quote and the application moves to underwriting
Member rejects quote

Basic course of action

1. Broker requests quote for product on behalf of the policyholder
2. Broker selects the product
3. System requests member details according to product specification and rules
4. Broker enters the required information and requests the monthly premium for a sum assured amount
5. System validates the information and returns a quote
6. If the policyholder accepts the quote, the quote is moved to the underwriters
7. The use case ends

Alternative Course A:

1. Member rejects the quote
2. The use case ends

Figure 6-7 Use case example for quoting business

6.3.1.3. Data warehouse user interface prototyping

In terms of OO, the user interface serves as the visual interface between the system and the user, whereas in DW the user interface is in the form of outputs from the data warehouse to a downstream system. Most of the use interfaces are based on the outputs of the different DW use cases.

The user interfaces that should be prototyped are:

- Reports (departmental)
- End user desktop applications
- Data mining models
- Downstream operational systems

Reports are typically information on business processes received by managers (or users) of the different departments, or generated from a reporting platform. This reporting platform is an end user system built on top of the data warehouse. The purpose is to model the report according to the needs of the manager or user of the department in question. A typical example of a report would be a sales report containing relevant elements, such as product name, date of sale, quantities of the product sold and the region where it is sold.

An end user desktop application only needs specified types of data. The department using the specific application should provide the requirements for that application. Data mining models also depend on specific data available, the correct data requirements to be provided by the mining modeller.

The same logic as with desktop application and mining models can be applied to the downstream application requirements. User interface prototyping should provide an idea of what information is needed by the different parties. After all the interfaces are prototyped, the domain modelling can start.

6.3.1.4. Domain Modelling

Domain modelling seeks to identify classes and objects common to all applications within the domain (Booch, 1991:157). Concepts (classes) are derived from nouns and noun phrases in the OO business essential use cases.

Class Responsibility Collaborator Cards (CRC) is a useful technique for discovering the classes that represent concepts. The technique, which should be used in a brainstorming session, is explained in Chapter 4.

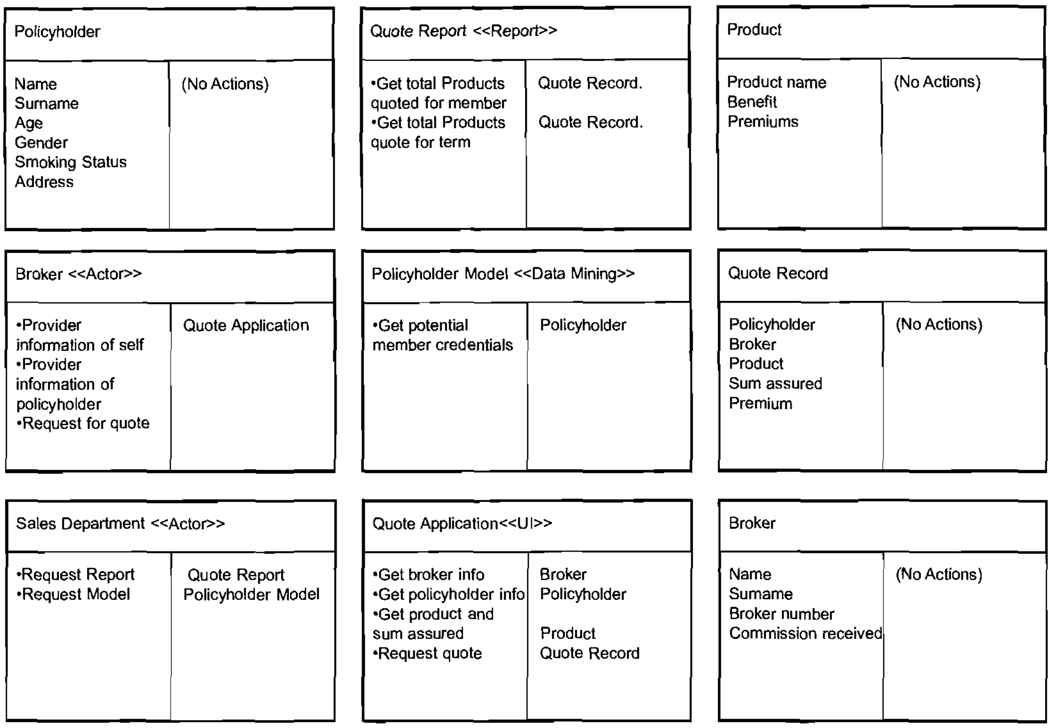
In normal OO development, three types of classes are created:

- Actor classes <<Actor>>
- Business classes
- User interface classes <<UI>>

The above types of classes can be used in conjunction with the following classes to model the DW domain:

- Reports <<Report>>
- End user desktop applications <<End user app>>
- Data mining models <<Data mining>>
- Downstream operational system interfaces <<Downstream Ops>>

Figure 6-8 provides an example of these types of classes. The CRC domain is called the DW CRC model in order to avoid confusion with the operational CRC domain model. Figure 6-8 is an example of the DW CRC cards involved in quoting for insurance.



Policyholder	
Name Surname Age Gender Smoking Status Address	(No Actions)

Quote Report <<Report>>	
•Get total Products quoted for member •Get total Products quote for term	Quote Record. Quote Record.

Product	
Product name Benefit Premiums	(No Actions)

Broker <<Actor>>	
•Provider information of self •Provider information of policyholder •Request for quote	Quote Application

Policyholder Model <<Data Mining>>	
•Get potential member credentials	Policyholder

Quote Record	
Policyholder Broker Product Sum assured Premium	(No Actions)

Sales Department <<Actor>>	
•Request Report •Request Model	Quote Report Policyholder Model

Quote Application<<UI>>	
•Get broker info •Get policyholder info •Get product and sum assured •Request quote	Broker Policyholder Product Quote Record

Broker	
Name Surname Broker number Commission received	(No Actions)

Figure 6-8 Example of a DW CRC model

Figure 6-8 illustrates the business classes as follows:

- Broker – contains the information about the broker
- Policyholder – contains the information about the policyholder
- Product – contains the information about the product
- Quote Record – contains the information about the actual quote

Responsibilities in the business classes are candidates for attributes in the class, which in turn is also a potential dimensional model.

The actor classes are:

- Broker – the individual who requests the quote
- Sales Department – the department who requests the reports and data mining model

The actor classes' responsibilities collaborate with the Quote Report, Policyholder model and the Quote Application classes. The actor cards initiate the process.

The user interface classes are:

- Quote Report of class report – the information displayed by the report
- Policyholder Model of class data mining model – the information needed by the model
- Quote Application of class user interface – the information displayed on the user interface

6.3.1.5. Supplementary Documentation

The specification documentation forms part of the requirements phase. The following is an overview of these documents, while a detailed discussion will follow at a later stage.

The specification documentation should contain the following:

- Business rules
- Outline of the data warehouse maintenance and growth
- Initial project plan
- Outline of the technical architecture design

Business rules

Business rules documentation contains the rules for the business, as well as for the product. Table 6-3 illustrates the rules that are specified in Figure 6-7.

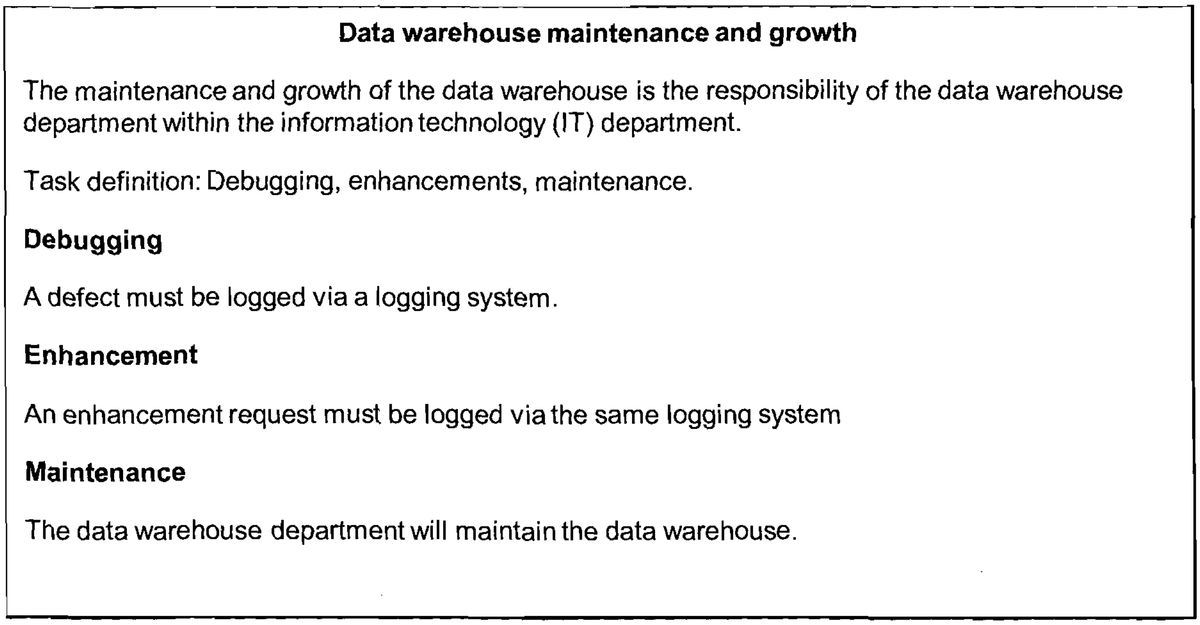
Rule no	***Description***
BR01	Policyholder must be older than 21 years
BR02	Policyholder must be female for maternity products
BR03	Policyholder can only have one life assurance product
BR04	Policyholder must be quoted via an accredited broker

Table 6-3 Business rules

The business rules serve as a base on which the Extract-Transform-Load (ETL) process will be built.

Outline of the data warehouse maintenance and growth.

The document on data warehouse maintenance and growth should specify the department or group responsible for the maintenance and development of the data warehouse. The document should also specify the formal procedure for maintaining change control (Figure 6-9).



Data warehouse maintenance and growth

The maintenance and growth of the data warehouse is the responsibility of the data warehouse department within the information technology (IT) department.

Task definition: Debugging, enhancements, maintenance.

Debugging

A defect must be logged via a logging system.

Enhancement

An enhancement request must be logged via the same logging system

Maintenance

The data warehouse department will maintain the data warehouse.

Figure 6-9 Documentation on data warehouse maintenance and growth

Initial project plan.

The initial project plan should contain the tasks needed by the development- and infrastructure teams, as well as estimations for these tasks.

Outline of the technical architecture design.

This documentation should contain the initial requirements needed by the technical architecture design team. It should contain detail of the following:

- Data staging
- Deployment

Data Staging

Documentation on the software and the source systems available are needed. For example, the database management system will be written in Oracle on a UNIX platform. The extract, transform and load tool will be a tool called Data Stage.

The available sources for data staging are:

- Policy administration application
- Billing and payment application
- Policy quote application.

Deployment

Physical requirements will include the procedure for deployment and a description of the environments. The requirements gathering phase is an iterative phase; as development proceeds, new requirements are discovered and documentation amended accordingly.

6.3.1.6. Requirements validation

Once the requirements are gathered, it should be validated. This can be done using the following techniques (discussed in chapter 4):

- Use case scenario testing
- User interface walkthroughs
- Requirements reviews

After validation of the requirements, the analysis of technical architecture, dimensional modelling and end user applications can be done.

The section on DW requirements concludes with a discussion on the suitability of OO techniques for DW requirements.

6.3.1.7. The Data Warehouse requirements based on OO approach

In terms of the requirements phase, the difference between the traditional Kimball *et al.* (1998) data warehouse development and the Object-Oriented Data Warehouse (OODW) is the following:

- OODW uses the technique of a normal OO essential use case diagram and the DW use case diagram to provide the detail, as well as the bigger picture of the company.
- OODW uses the technique of a normal OO essential use case and the DW essential use case which explain the inputs, outputs and the grain of the department.
- OODW uses the technique of user interface prototyping to form ideas of what is needed from the data warehouse in terms of DW outputs.
- OODW uses the technique of domain modelling to define the different objects and classes that will be used in the DW.

Traditional Kimball *et al.* (1998) DW development does not use any of the above techniques for gathering requirements.

6.3.2. Dimensional modeling

Dimensional modelling is defined as "a logical design technique that seeks to present the data in a standard framework that is intuitive and allows for high performance access. It is inherently dimensional and adheres to a discipline that uses relational model with some important restrictions" (Kimball *et al.*, 1998:144).

The dimensional modelling phase should allow for an analysis and design activity in order to follow a typical OO development approach.

The discussion will concentrate firstly on the analysis of the dimensional modelling and thereafter on the design thereof.

Figure 6-10 illustrates an outline of the activities in OO analysis.

Phase	Activity
Object Oriented Analysis	• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages

Figure 6-10 Object Oriented Analysis activities

6.3.2.1. Dimensional modelling analysis – DW System use case

Up to this point two types of use cases were developed, namely:

- Business process essential use cases
- Data warehouse essential use cases

As with essential use cases, the system use case is also divided into a business process system use case and a DW system use case. Both the business and DW system use cases are evolved from their respective essential use cases.

The DW system use case follows the same pattern as the DW essential use case, except that it entails more specific information with regard to inputs, outputs and the grain of the department. It also specifies the users and their requirements.

Figure 6-11 is an example of a DW system use case for the sales department.

Name: DWUC 02 Sales

Description: Sales department is responsible for the sales of a product

Basic inputs

Sales staff should provide the following:

1. 1st line managers provide sales figures via sales_mm_dd_yyyy.mdb file on network drive
2. sales_mm_dd_yyyy.mdb contains the following information:
    a. Sales of each product (unit in each)
    b. Dates of sold product (format dd-mm-yyyy)
    c. Regions of sold products
    d. Brokers responsible for sales in regions (format: Title Name Surname)

Basic outputs

1. Regional sales figures (RPT_S01)
2. Broker sales quota (RPT_S02)
3. Quarter sales (RPT_S03)

Possible grain(s)

1. The sale of a product

Users

1. Product modelling manager requires the following – (RPT_S01, RPT_S03)
2. Sales manager requires the following – (RPT_S01,RPT_S02,RPT_S03)
3. Finance manager requires the following – (RPT_S01, RPT_S03)

Figure 6-11 Data warehouse system use case example for sales department

6.3.2.2. Dimensional modelling analysis – Business process system use case

The business process system use case is similar to the business process essential use case with the exception that it includes high-level implementation decisions, such as screen numbers and properties, such as includes, extends and inherits.

Figure 6-12 is an example of a business process system use case

Name: Quote for insurance

Identifier: UC01

Description: Broker requests a quote for insurance policy for a prospective policy holder

Preconditions:

Post conditions:

Member accepts the quote and the application moves to underwriting

Member rejects the quote

Extends: –

Includes: –

Inherits from: –

Basic course of action

1. Broker requests quote for product on behalf of the policyholder via "*UI01 Request quote for insurance*"
2. System requests member details according to product specification and rules according business rules BR01 – Policyholder must be older than 21 years
3. Broker enters the required information and requests the monthly premium for a sum assured amount
4. System validates the information and returns a quote with the business rules BR02 – BR04
5. If the policyholder accepts the quote, the quote is moved to the underwriters
6. The use case ends

Alternative Course A:

3. Member rejects the quote
4. The use case ends

Figure 6-12 Business process system use case example for quoting business

6.3.2.3. Dimensional modelling analysis – Sequence diagrams

Sequence diagrams are developed from use cases. Jacobson *et al.* (2001:251) states that the function of sequence diagrams is to model the logic of usage scenarios. A usage scenario is a description of a potential way the system can be used; this may include use cases or alternative courses. It provides a bridge between the business system use cases and the class models.

Figure 6-13 is an example of a sequence diagram derived from the business process system use case in Figure 6-12.

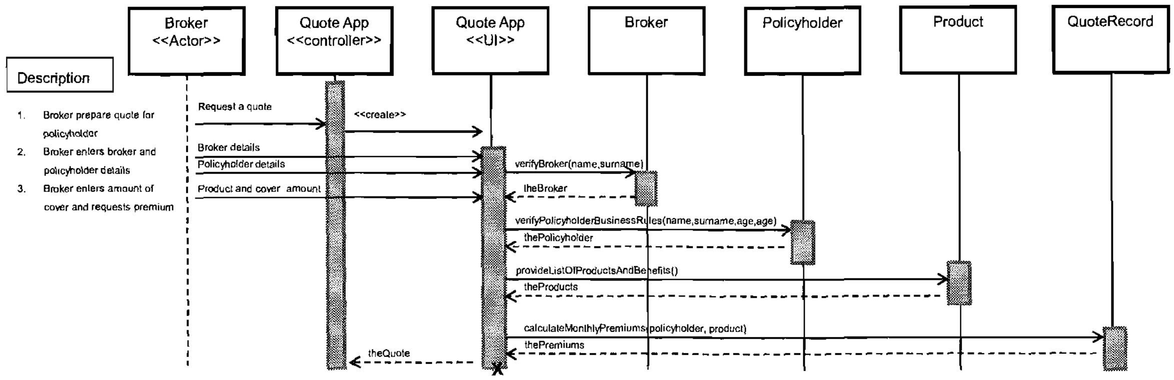

Broker <<Actor>>
Quote App <<controller>>
Quote App <<UI>>
Broker
Policyholder
Product
QuoteRecord
Description
1. Broker prepare quote for policyholder
2. Broker enters broker and policyholder details
3. Broker enters amount of cover and requests premium
Request a quote
<<create>>
Broker details
Policyholder details
Product and cover amount
verifyBroker(name,surname)
theBroker
verifyPolicyholderBusinessRules(name,surname,age,age)
thePolicyholder
provideListOfProductsAndBenefits()
theProducts
calculateMonthlyPremiums(policyholder, product)
thePremiums
theQuote

Figure 6-13 Sequence diagram for quoting for insurance

In terms of interaction, the sequence diagram (Figure 6-13) illustrates that there is an interaction between:

- The broker
- The policyholder
- The policy
- The quote record

The significance of this interaction will be explained in the following discussion.

The function of the data warehouse use cases is to provide an overview of the "big picture" of the company as opposed to the process flow in the business. It therefore will be senseless to create a sequence diagram from the data warehouse use cases, as the function of a sequence diagram is to model the interaction between classes in a process.

For purpose of the dimensional modelling, the logic flow is not all that important, but rather important to discover which classes interact with one another and what that interaction entails.

6.3.2.4. Dimensional modelling analysis – The Data Warehouse Bus Architecture Matrix

The Data Warehouse Bus Architecture Matrix is a tool introduced by Kimball (*et al.*, 1998:271) to decide which dimensional models to build. The matrix forces to name all the data marts that can possibly be built, as well as the dimensions implied by those data marts.

The business process system use case contains the concepts entailed in the business, thus giving an indication of the possible data marts and dimensions on hand.

For the purpose of creating a data warehouse bus matrix, the sequence diagram serves as an indicator of which classes interact with one another.

The following discussion illustrates how to create a data warehouse bus matrix from the DW system use case, business process system use case and the sequence diagram.

Identifying the data marts and dimensions

The department column in Table 6-1 lists the potential data marts that can be used for the matrix. In some cases the department name can be too encompassing and should be divided into smaller units. This should be highlighted by the business essential use case in its business processes.

To gain a better understanding of the company, a table should be created combining the DW use cases with the business use cases. Table 6-4 is an example of such a table.

DW Use Case Number	Department	Business Use Case Number	Business Process
DWUC01	Product modelling		
DWUC02	Sales	UC 01	Quote
DWUC03	Marketing		
DWUC04	Claims	UC05	Claim on policy
		UC06	Assessment of policy
DWUC05	Finance	UC02	Commission paid to broker
DWUC06	Billing	UC 04	Billing of policy premiums

Table 6-4 Combination of the DW use cases with the business use cases

From Table 6-4, the following can be derived:

- DWUC01 does not contain any business use cases that match the DW use case. One can therefore do a future analysis on this department to be included in the DW, or it can be left out of the development scope of the DW.
- DWUC02 contains only the quoting business process.
- DWUC03 is similar to DWUC01; it can be investigated further of left out for now.
- DWUC04 contains two business processes, namely claiming against the policy (UC05) and assessment of the policy (UC06). These business processes can be grouped together in the data mart.
- DWUC05 contains only the commission business process.
- DWUC06 contains only the billing business process.

The conclusions above will serve as the subject areas in the data warehouse matrix.

The next step is to identify the dimensions of the data warehouse matrix. The following example illustrates the evaluation process for finding dimensions. The business classes identified in the insurance company example are:

- Policyholder
- Product
- Quote Record

- Broker

The actor classes are:

- Broker
- Sales Department

The user interface models are:

- Quote Report <<Report>>
- Policyholder Model <<Data Mining>>
- Quote Application <<UI>>

These classes are potential dimensions for the data warehouse. The sequence diagram created in Figure 6-13 illustrates the sequence of tasks for the quoting process. It interacts with the following classes:

- Policyholder
- Product
- Quote Record
- Broker
- Broker <<Actor>>
- Sales Department <<Actor>>
- Quote Report <<UI>>
- Policyholder Model <<UI>>

Each class describing the process can qualify as a dimension. These classes should also be specified in the sequencing diagram. The following classes qualify as dimensions:

- Policyholder – describes the policyholder in terms of the quote process
- Product – describes the policyholder in terms of the quote process
- Broker – describes the policyholder in terms of the quote process

The following classes do not qualify as dimensions:

- Quote Record – does not qualify as it does not tie the classes together; it does not describe the quote process implicitly or data mart
- Broker <<Actor>> – does not qualify as it does not identify the processes and does not describe the quote process or data mart implicitly
- Sales Department <<Actor>> – does not qualify as it does not describe the quote process or data mart
- Policyholder Model <<Actor>> – does not qualify as it does not describe the quote process or data mart

An evaluation process should be completed for every process or data mart. A time dimension, indicating when a certain process occurred, should be added.

A data warehouse bus matrix is created with the data marts and dimensions derived from the evaluation process. The dimensions are listed as columns and the data marts as rows.

Figure 6-14 is an example of a data warehouse matrix created for the insurance company.

	Time	Policyholder	Broker	Product	Claim Description	Underwriter
Quote	√	√	√	√		
Underwrite	√	√	√			√
Commissions	√	√	√	√		
Billing	√	√		√		
Claims	√	√		√	√	

Figure 6-14 The Data Warehouse Bus Architecture matrix for the insurance company example.

As previously discussed, the sequence diagram indicates which processes interact with which classes. This information can be used to derive which data marts interact with which dimensions. The interaction is indicated by the tick in the matrix.

Once all the potential data marts and dimensions are identified, the four-step method of Kimball *et al.* (1998:273) can be applied to develop dimensional models. The theory on the four-step method for designing an individual fact table was discussed in chapter 5 (Section 5.1.2.1).

The following illustrates the four-step method applied for one data mart.

Step 1: Choose the Data Mart

The data mart selected is the quote data mart. For the insurance company example, the quote data mart is selected.

Step 2: Declare the Grain

The grain specifies the level of detail of the model; the grain for the dimensional model will be a one record per quote transaction. This is defined as a record with details of the broker, policyholder, the product quoted on, the sum assured and the premiums.

Step 3: Choose the Dimensions

The data mart selected in the data warehouse bus architecture matrix will indicate which dimensions can be used. According to Figure 5-6 the quote data mart should have the following dimensions:

- Time
- Policyholder
- Broker
- Product

Step 4: Choose the facts

Facts in the fact table are usually numerical values. The numerical values in the quote record, in Figure 6-8 is the premiums and the sum assured values. The numerical value in the attributes of the classes in the CRC diagram serves as a starting point, as many facts should be generated within the context of the grain. Based on the four-step method, a dimensional model can be developed.

At this stage it is advisable to do an analysis on the technical architecture, the reason being that the dimensional model and technical design closely correlate with one another. A discussion on the analysis of the technical architecture will follow after design of the dimensional models has been dealt with.

6.3.2.5. Dimensional modelling analysis – Dimension table detail

The dimension table diagram needs to be completed for each single dimension. It illustrates the grain of each dimension, as well as the cardinality of each dimension attribute, with a top down view of all the hierarchies (Kimball *et al.*, 1998:281). Figure 6-15 is an example of a dimension table detail diagram for the time dimension.

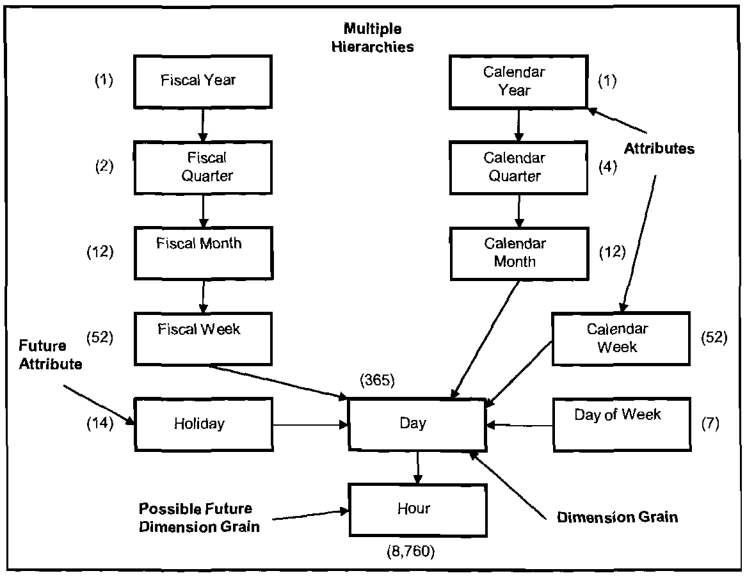
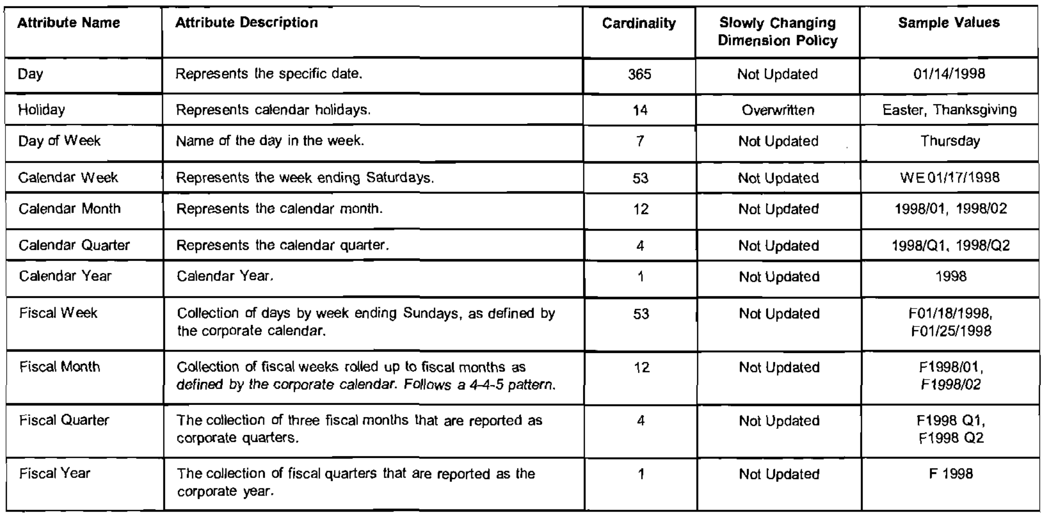


Figure 6-15 Dimension table diagram (Kimball *et al.*, 1998:281)

In Figure 6-15, the rectangles represent the attributes specified for the dimension, while the cardinality is shown in parentheses. The arrows between the rectangles represent the drill paths for the hierarchy. Post-dated attributes can also be specified on the diagram. In the above case, one can follow the hierarchy Fiscal Year → Fiscal Quarter → Fiscal Month → Fiscal Week → Day, or the hierarchy Calendar Year → Calendar Quarter → Calendar Month → Day.

Alongside the diagram is the attribute detail description. Figure 6-16 is an example of an attribute detail description for the time dimension table diagram.



Attribute Name	Attribute Description	Cardinality	Slowly Changing Dimension Policy	Sample Values
Day	Represents the specific date.	365	Not Updated	01/14/1998
Holiday	Represents calendar holidays.	14	Overwritten	Easter, Thanksgiving
Day of Week	Name of the day in the week.	7	Not Updated	Thursday
Calendar Week	Represents the week ending Saturdays.	53	Not Updated	WE01/17/1998
Calendar Month	Represents the calendar month.	12	Not Updated	1998/01, 1998/02
Calendar Quarter	Represents the calendar quarter.	4	Not Updated	1998/Q1, 1998/Q2
Calendar Year	Calendar Year.	1	Not Updated	1998
Fiscal Week	Collection of days by week ending Sundays, as defined by the corporate calendar.	53	Not Updated	F01/18/1998, F01/25/1998
Fiscal Month	Collection of fiscal weeks rolled up to fiscal months as *defined by the corporate calendar. Follows a 4-4-5 pattern.*	12	Not Updated	F1998/01, *F1998/02*
Fiscal Quarter	The collection of three fiscal months that are reported as corporate quarters.	4	Not Updated	F1998 Q1, F1998 Q2
Fiscal Year	The collection of fiscal quarters that are reported as the corporate year.	1	Not Updated	F 1998

Figure 6-16 Dimension attribute detail description (Kimball *et al.*, 1998:283)

The attribute detail description consists of five columns:

- Attribute name – official name of the attribute.
- Attribute description – a description of the attribute.
- Cardinality – an estimation of the distinct values of the attribute.
- Slowly changing policy – is the type of slowly changing attribute, e.g. Type 1 – overwritten, Type 2 – new version of the attribute and Type 0 – the value is never updated.
- Sample data – Sample value of the attribute.

6.3.2.6. Dimensional modelling analysis – Fact table diagram

Figure 6-17 is an example of a fact table diagram for the quote fact. The fact table diagram illustrates the specific fact table within its own context and also serves as an overview of all the dimensions that have been identified. The names and descriptions of these dimensions are shown (Kimball *et al.*, 1998:277).

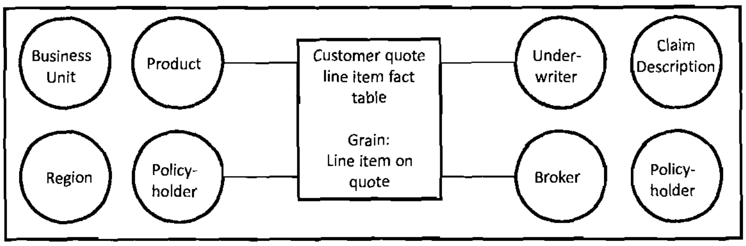
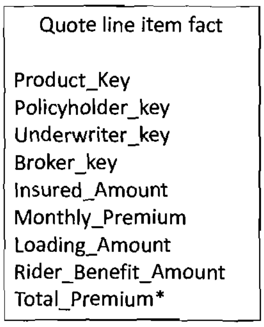


Figure 6-17 Quote fact table diagram

All dimensions are illustrated in the fact table diagram. Only those dimension tables that interact with the fact table are connected to the latter, the others not.

Once the fact table diagrams are done for each fact, the fact table details can be created. Figure 6-18 illustrates the fact table detail for quote; it includes the keys and the facts. The fact with an asterisk represents a derived fact.



Quote line item fact

Product_Key
Policyholder_key
Underwriter_key
Broker_key
Insured_Amount
Monthly_Premium
Loading_Amount
Rider_Benefit_Amount
Total_Premium*

Figure 6-18 Fact table detail for quote fact table.

The following illustrates how the attributes are generated for fact table detail (Figure 6-18).

Product_key → derived from the fact table diagram (key for product dimension)

Policyholder_key → derived from the fact table diagram (key for policyholder dimension)

Underwriter_key → derived from the fact table diagram (key for underwriter dimension)

Broker_key → derived from the fact table diagram (key for broker dimension)

Insured_Amount → derived from DW CRC model (Sum Assured in Figure 6-8 quote record)

Monthly_Premium → derived from DW CRC model (Premium in Figure 6-8 quote record)

Loading → derived from DW CRC model (Premium in Figure 6-8 quote record)

Rider_Benefit_Amount → derived from DW CRC model (Premium in Figure 6-8 quote record)

Total_Premium is a derived premium that sums the monthly loading and rider benefit premiums.

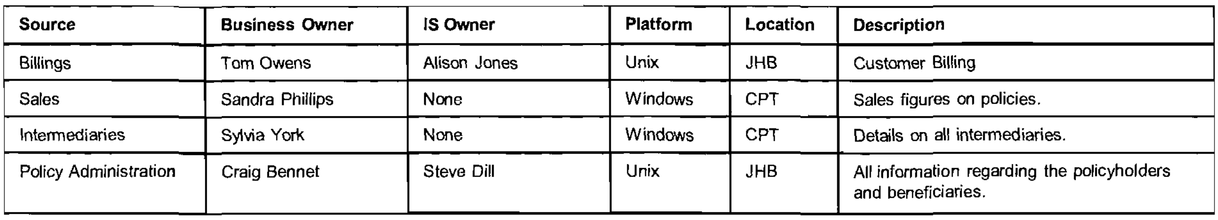
6.3.2.7. Dimensional modelling analysis – Identify sources

Two types of data sources need to be identified (Kimball *et al.*, 1998:296):

- Informal sources – Data captured on a user's database
- Formal sources – Data maintained by IS

Both sources are subject to a process of cleaning and manipulation before they can be stored in the data warehouse. A data source list also needs to be created.

Figure 6-19 is an example of such a source list.



Source	Business Owner	IS Owner	Platform	Location	Description
Billings	Tom Owens	Alison Jones	Unix	JHB	Customer Billing
Sales	Sandra Phillips	None	Windows	CPT	Sales figures on policies.
Intermediaries	Sylvia York	None	Windows	CPT	Details on all intermediaries.
Policy Administration	Craig Bennet	Steve Dill	Unix	JHB	All information regarding the policyholders and beneficiaries.

Figure 6-19 Data source definition

The data source list contains the following (Kimball *et al.*, 1998:298):

- Source – name of the source system
- Business owner – the primary contact person responsible for this information
- IS Owner – the contact person responsible for the source system
- Platform – the operating system on which the system runs
- Location – the location of the system
- Description – a brief description of the system

Once the data source definition list is available, it can be investigated and analysed. From this, a source-to-target mapping is created, indicating which fields in what sources need to go to which dimension table field.Figure 6-20 is a source-to-target data map example for the period, product and quote dimensions.

Table Name	Column Name	Data Type	Len	Target Column Description	Source System	Source Table / File	Source Column / Field	Data Transform
Period Dimension	PERIOD_KEY	Date	-	The unique primary key for the period dimension table	New	New	New	New
Product Dimension	PROD_KEY	Int	8	The unique primary key for the product dimension table.	PA	PROD_INFO	SH_ID	Get member Number
Product Dimension	PROD_DESC	CHAR	255	Description of the product	PA	PROD_INFO	DESC	Direct
Quote Fact	QUOTE_KEY	Int	-	The unique primary key for each quote generated.	PQS	QUOTE_INFO	QUOTE_ID	Direct
Quote Fact	QUOTE_DATE	Date	-	The date that the quote was generated	PQS	QUOTE_INFO	Q_DATE	Direct

Figure 6-20 Source-to-target data map

The source-to-target data map contains the following columns (Kimball *et al.*, 1998:305):

- Table name – the name of the logical table in the data warehouse.
- Column name – the name of the logical column in the data warehouse.
- Data type – the data type of the column in the data warehouse.
- Length – the length of the field of the column.
- Target column description – a description of the target column.
- Source table / file – the name of the source system where data feeds the target column.
- Source column / field – the name of the specific column within the source table where the data feeds from.
- Data transform – any information needed to translate the source information to the format of the target system.

6.3.2.8. Dimensional modelling design – Development of dimensional tables

The development of the dimensional tables requires the following analysis documents:

- Data warehouse matrix – illustration of the data marts and the dimensions available for the specific data mart.
- Fact table diagram – illustration of the fact table detail within its context.
- Dimensional table detail – illustration of the hierarchies in the dimension tables.
- Sources detail – a list of available source data and the owners of the data.
- Source-to-target mapping – mapping from the source data to the target dimensional tables.

On completion of the documents, a dimensional model can be created, as illustrated in Figure 6-21.

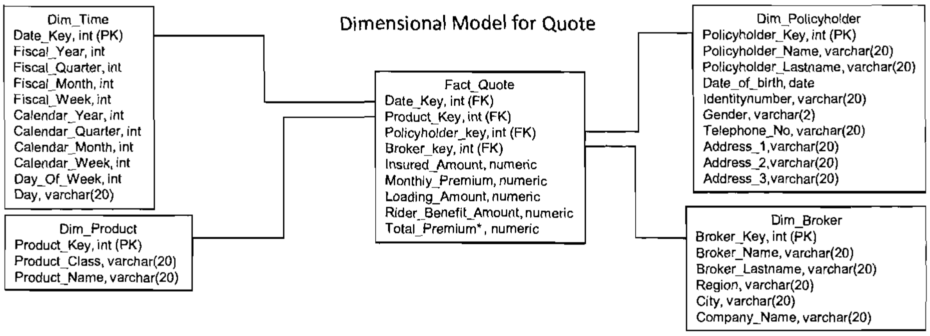


Figure 6-21 Quote dimensional model

Figure 6-21 is a dimensional model for the quote data mart and contains four dimensions with the attributes (being the design of the dimensional model). The dimensions created for the quote fact table corresponds with the ticked data warehouse matrix (Figure 6-14). A dimensional model for each data mart listed in the data warehouse matrix should be created.

Kimball *et al.* (1998:309) recommends that a modelling tool should be used to develop the data model, the reasons for this being:

- Consistency in naming.
- Documentation can be created from the objects.

- Generation of the physical data definition language (DDL).
- Supportive user interface.

This section on DW modelling concludes with a discussion on the applicability of OO techniques for DW modelling.

6.3.2.9. The Data Warehouse dimensional modelling based on an OO approach

In terms of the dimensional phase in data warehouse development, the difference between the traditional Kimball *et al.* (1998) data warehouse development and the OO Data Warehouse (OODW) is the following:

- OODW splits the dimensional modelling into two sub-phases, namely the analysis of the dimensional model and the design of the dimensional model.
- In the analysis phase of the OODW dimensional model, the DW essential use cases and the business essential uses cases are combined to obtain the potential subject areas for the DW. Traditional Kimball DW development does not use a technique to derive the subject areas. Instead, it analyses the business processes and based on these results, they are grouped into subject areas.
- In the analysis phase of the OODW dimensional model, the selected essential use case classes are used as the dimensions. These are combined with the subject areas to create the DW matrix. Traditional Kimball DW development does not use a technique to derive the dimensions.
- OODW uses the same design techniques than for traditional Kimball DW development, as it is advisable to adhere to traditional DW designs in line with industry standards.

6.3.3. Technical Architecture modelling

Technical architecture modelling comprises an analysis and design phase. Based on the OO activities listed in Figure 6-1, the activities shown in Figure 6-22 will be carried out.

Phase	Activity
Object Oriented Analysis	• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages

Phase	Activity
Object Oriented Design	• Class Modelling • State Chart Modelling • Collaboration Modelling • Component Modelling • Deployment Modelling • Relational Persistence Modelling • User Interface Design

Figure 6-22 Object Oriented analysis and design

Figure 6-23 illustrates a high-level technical architecture of a typical data warehouse.

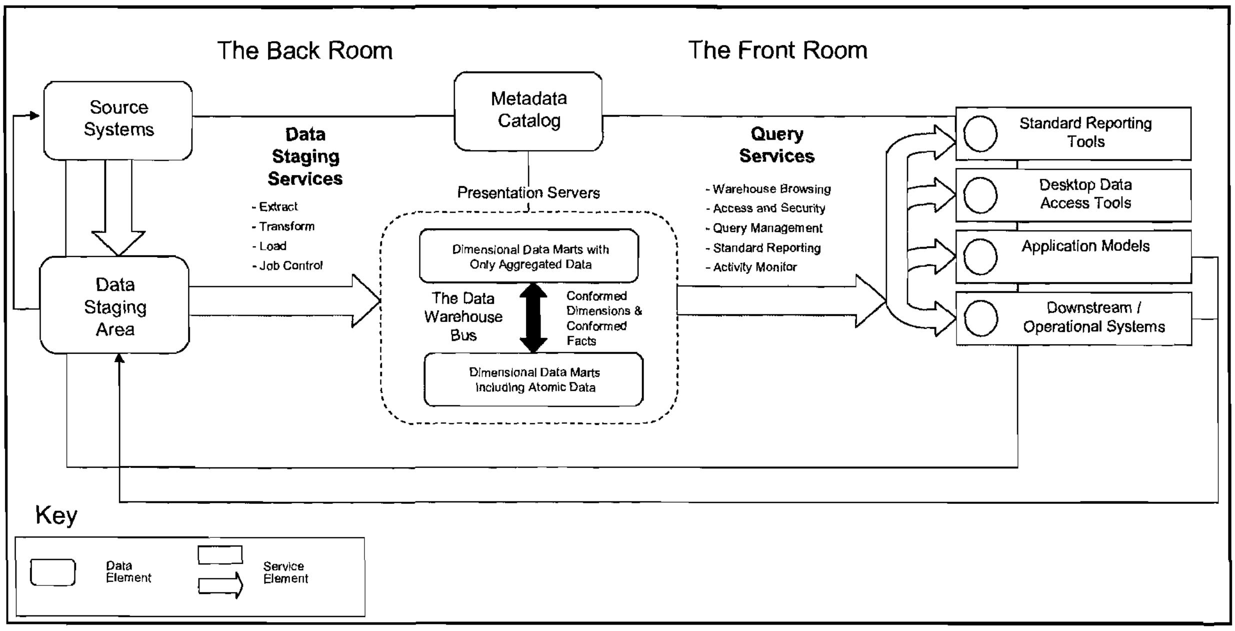


Figure 6-23 High-level technical architecture model (Kimball *et al.*, 1998:329)

The model provides for a logical separation between the internal working of the warehouse and the user front end (Kimball *et al.*, 1998:329). The analysis and design of such architecture should therefore be separated according to the back and front room of the model.

The analysis and design of the back room entail the following activities:

- Source system analysis
- Data staging services analysis
- Data staging services design

The analysis and design of the front room consist mainly of the query services.

6.3.3.1. Technical Architecture back room OO analysis – Source systems

The analysis needed for the architecture is done during the identification of sources for the dimensional models. The sources of the different dimensional models should be consolidated and used in the data staging area development.

6.3.3.2. Technical Architecture back room OO analysis – Data staging services

The data staging services consist mainly of the services listed below. The OO development of these services is discussed in section 6.3.5.

- Extract
- Transform
- Load
- Job control

Extract

The analysis of the extraction of data requires the following analysis documents (Kimball & Caserta, 2004:55):

- Source-to-target mapping for all the dimensions
- Entity relational (ER) model of the source data
- Business rules that influence the Extract-Transform-Load (ETL) process

The source-to-target mapping is done during the dimensional model analysis. This list should be consolidated into a master list. The entity relationship model of the source systems must be obtained and analysed, in order to determine how the data will be extracted from the database. However, data can also be extracted from flat files.

The purpose of the source system ER model is to understand what that source data looks like and to determine what the system of record is. Kimball and Caserta, (2004:66) simply defines the system of record as the originating source of data. Figure 6-24 is an example of an ER model of a database.

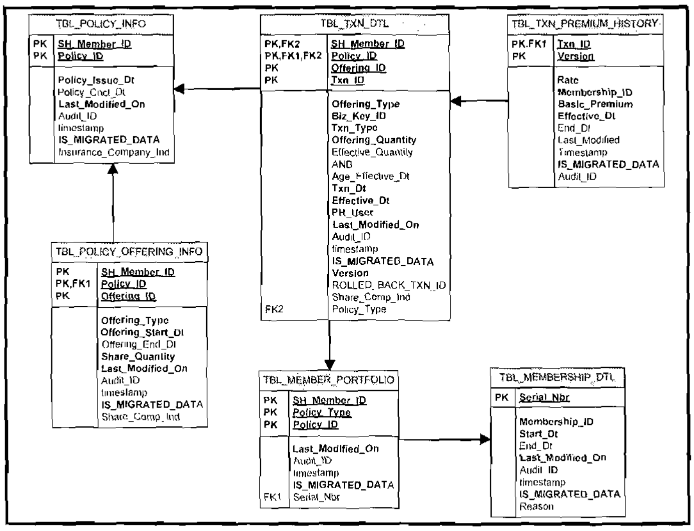


Figure 6-24 Entity Relationship model of a sample database

Based on the ER model in Figure 6-24, the source of record is the policy offering table. The beginning of the dataflow starts with the policy entry.

During the requirements phase, the business rules are defined. The rules identified should be investigated, as this can have an influence on the formatting of data. Table 6-5 is an example of such rules.

Rule no	Description
BR05	Dates are in the format of YYYY-MM-DD.
BR06	The member has a stakeholder ID that is converted to a member number.

Table 6-5 Technical Business rules

Kimball and Caserta (2004:63) uses a so-called “source system tracking report” to list the different source systems, the function of the source data and the parties involved. A similar report is generated during the identification of a data source in dimensional modelling analysis.

Transform

Transformation requires two types of documents:

- Basic high level data stage schema plan
- Detailed plan
- List of derived facts
- List for changing dimensions

The basic level data stage schema plan illustrated in Figure 6-25, shows the sources (top) and the targets (bottom). The lines between the sources and the targets represent transformations. The major issues associated with a transformation are described in the rectangle on the line.

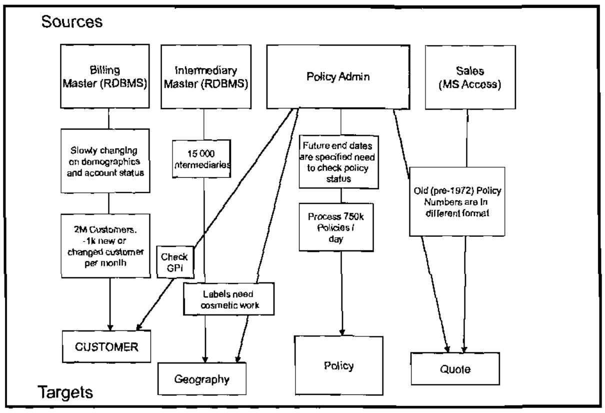

Sources
Billing Master (RDBMS)
Intermediary Master (RDBMS)
Policy Admin
Sales (MS Access)
Slowly changing on demographics and account status
15 000 ntermediaries
Future end dates are specified need to check policy status
Old (pre-1972) Policy Numbers are In different format
Process 750k Policies / day
2M Customers. -1k new or changed customer per month
Check GPI
Labels need cosmetic work
CUSTOMER
Geography
Policy
Quote
Targets

Figure 6-25 High level data stage schema plan

A detail plan should be created for each flow of data (from source to target) in the high level plan. Figure 6-26 is an illustration of such a detail plan. The source is shown in the top left hand corner, and the major transformation issues are listed as indicated by the arrows. In the bottom right hand corner, the final fact /dimension table, to where the data should be loaded, is shown. The tests on the lines contain critical processing information.

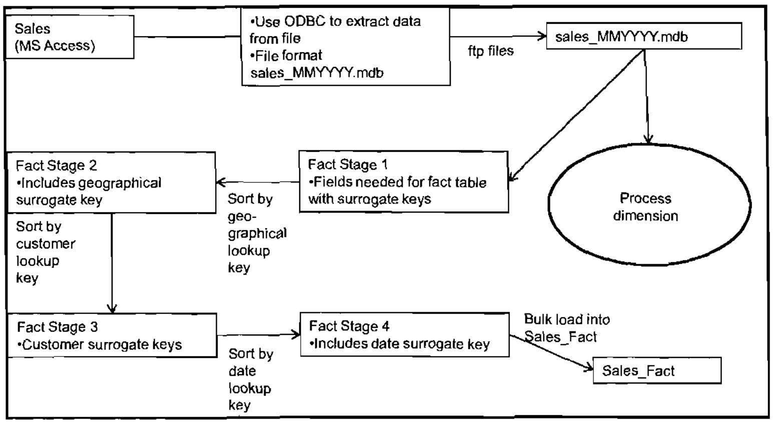

Sales
(MS Access)
•Use ODBC to extract data from file
•File format sales_MMYYYY.mdb
ftp files
sales_MMYYYY.mdb
Fact Stage 2
•Includes geographical surrogate key
Fact Stage 1
•Fields needed for fact table with surrogate keys
Process dimension
Sort by geo-graphical lookup key
Sort by customer lookup key
Fact Stage 3
•Customer surrogate keys
Fact Stage 4
•Includes date surrogate key
Sort by date lookup key
Bulk load into Sales_Fact
Sales_Fact

Figure 6-26 Detail schematic plan for fact table load

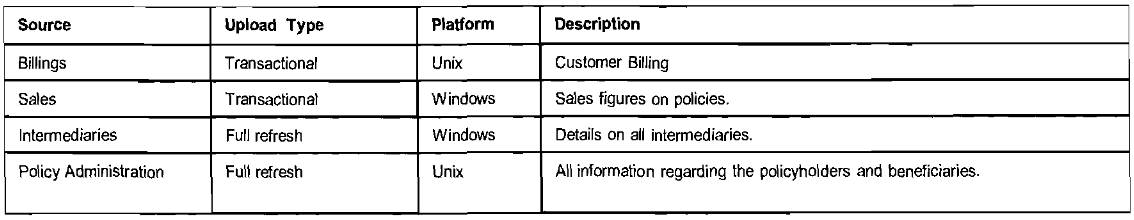
The list of derived facts is gathered during the analysis of the dimensional tables. The same is done for the list of slowly changing dimensions, also gathered during the dimensional table analysis. Both these lists should be consolidated for the design of the extract.

Load

For loading of data into the data warehouse, there are three kinds of loading processes (Kimball *et al.*, 2000:358):

- Incremental loads – involves processing monthly snapshots of the source system and uploading this information.
- Transaction events – involves processing the transactions one by one as it happens.
- Full refresh – involves taking the whole source database, processing and uploading it.

The source definition in conjunction with the dimensional table should provide the analysis with the type of upload needed. A list of sources and types of upload should be created (Figure 6-27).



Source	Upload Type	Platform	Description
Billings	Transactional	Unix	Customer Billing
Sales	Transactional	Windows	Sales figures on policies.
Intermediaries	Full refresh	Windows	Details on all intermediaries.
Policy Administration	Full refresh	Unix	All information regarding the policyholders and beneficiaries.

Figure 6-27 List of sources with the upload type

Job control

The job control services of the ETL process ensures that it is properly managed. Kimball *et al.* (1998:364) recommends that it should include the following:

- Job definition – definition of the series of steps needed to perform the job.
- Job scheduling – scheduling of the job should be done. This can be time or event based.
- Monitoring – ways to monitor the system while the ETL process is in progress.
- Logging – ways to collect information about the entire ETL process.
- Exception handling – ways to determine whether some of the processes failed.

6.3.3.3. Technical Architecture back room OO design – Data staging services

The analysis documents (gathered for the back room architecture) give the developer clear instructions as to how this architecture should perform. The design can be done by employing integrated ETL design tools, which will be discussed in section 6.3.5.

6.3.3.4. Technical Architecture front room OO analysis – Query services

The front room is a vital part of the model, as this is the part the users see and use to access the data warehouse (Kimball *et al.*, 1998:409). The types of users and interfaces are determined during the user interface prototyping and grouped into use cases. These use cases should be analysed to determine the common front end interfaces.

6.3.3.5. Technical Architecture front room OO design – Query services

The design of the front room architecture should be of such a nature that it supports the front end application. An appropriate development methodology therefore would depend on the nature of the front end. For example, if the front end is a customer desktop utility that is being developed, the design should follow an OO methodology.

This section on DW technical architecture concludes with a discussion on the applicability of OO techniques to DW technical architecture.

6.3.3.6. The Data Warehouse technical architecture based on an OO approach

In terms of the technical architecture phase in data warehouse development, the differences between the traditional Kimball *et al.* (1998) data warehouse development and the Object Oriented Data Warehouse (OODW) are the following:

- OODW splits the modelling of each of the services in the technical architecture into two sub-phases, namely analysis and design.
- The technical architecture model itself does not differ from the traditional Kimball model, but the method used to analyse and design the technical architecture differs. OODW uses an OO approach to the analysis and design of the technical architecture, whereas Kimball does not.

6.3.4. Physical Design

The physical design involves the design of the logical database, as well as its implementation. The process is as follows:

- Define naming standards
- Design physical tables and columns
- Estimate database size and index plan
- Develop aggregation plan

6.3.4.1. Physical Design – Define standards

A document explaining the naming standards for tables, attributes, synonyms and file locations should be created. The type of platform should also be documented. Examples of naming standards are displayed in Table 6-6.

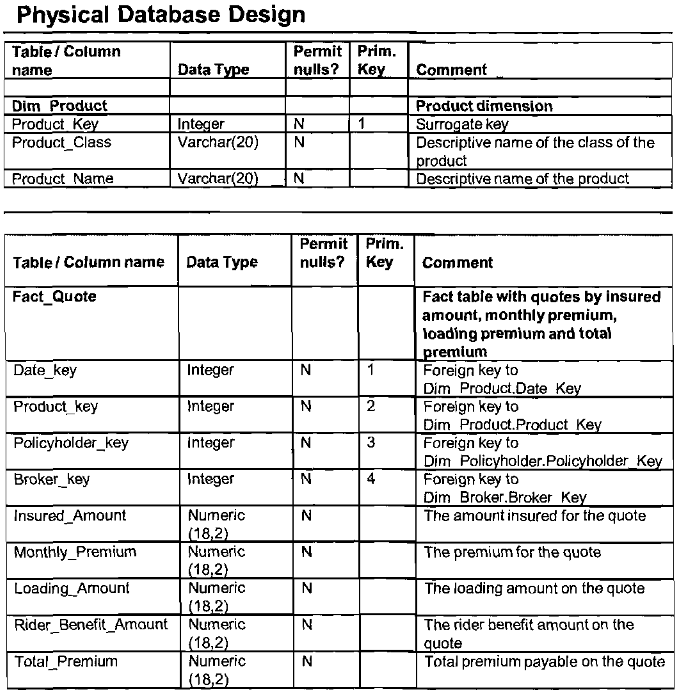
Standard No	Description
SR01	All dimensional tables should start with “Dim_”.
SR02	All fact tables should start with “Fact_”.

Table 6-6 Standards with descriptions

The design of the physical model should follow the standards document.

6.3.4.2. Physical Design – Design physical tables and columns

The dimensional table design serves as a logical model for the physical design. The physical design should be as close as possible to the logical model, except for the inclusion of the physical database specification and naming standards. Figure 6-28 is a physical data model based on the quote dimensional model.



Physical Database Design

Table / Column name	Data Type	Permit nulls?	Prim. Key	Comment
Dim_Product				**Product dimension**
Product_Key	Integer	N	1	Surrogate key
Product_Class	Varchar(20)	N		Descriptive name of the class of the product
Product_Name	Varchar(20)	N		Descriptive name of the product

Table / Column name	Data Type	Permit nulls?	Prim. Key	Comment
Fact_Quote				**Fact table with quotes by insured amount, monthly premium, loading premium and total premium**
Date_key	Integer	N	1	Foreign key to Dim_Product.Date_Key
Product_key	Integer	N	2	Foreign key to Dim_Product.Product_Key
Policyholder_key	Integer	N	3	Foreign key to Dim_Policyholder.Policyholder_Key
Broker_key	Integer	N	4	Foreign key to Dim_Broker.Broker_Key
Insured_Amount	Numeric (18,2)	N		The amount insured for the quote
Monthly_Premium	Numeric (18,2)	N		The premium for the quote
Loading_Amount	Numeric (18,2)	N		The loading amount on the quote
Rider_Benefit_Amount	Numeric (18,2)	N		The rider benefit amount on the quote
Total_Premium	Numeric (18,2)	N		Total premium payable on the quote

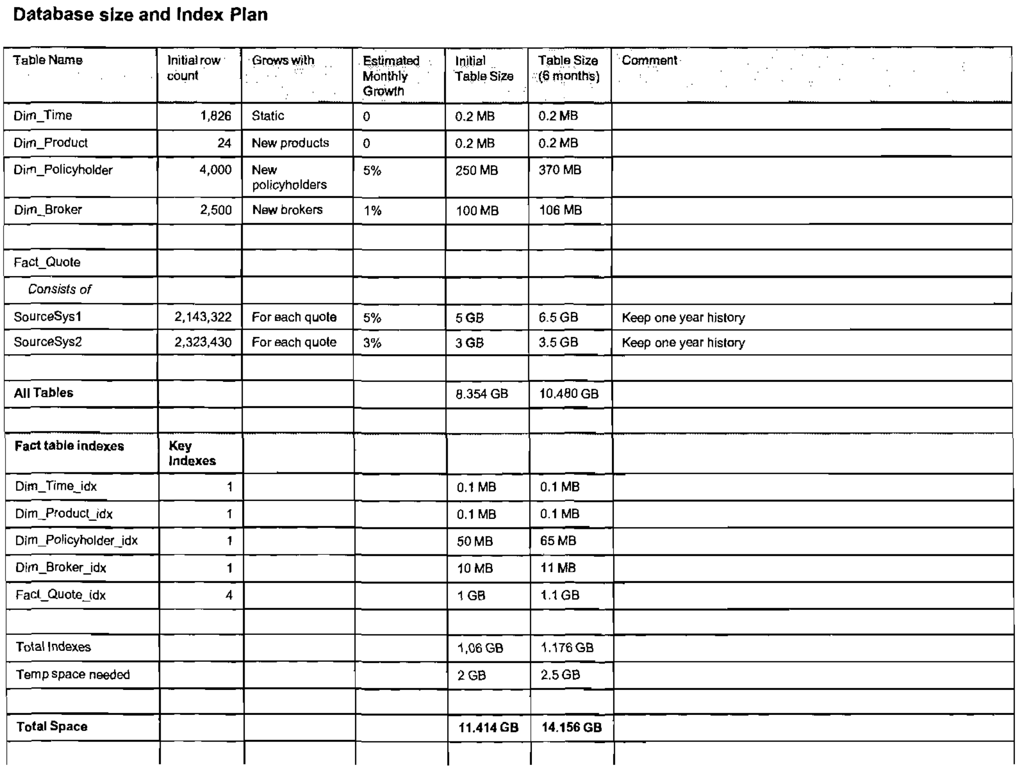
Figure 6-28 Partial physical model for quote

The naming standards should reflect in the physical model. Examples of these are the names of tables starting with “Dim” for dimension and “Fact” for fact, as well as the suffix “key” for key attributes.

The physical model contains the names of the attributes, the data types, null values and the combination of the primary key.

6.3.4.3. Physical Design – Estimate database size and index plan

An estimation of the correct database size is not easy and should be done by a qualified database administrator (DBA). Indexing of the table should also be done by the DBA. Thereafter, the DBA should produce a plan similar to Figure 6-29.



Database size and Index Plan

Table Name	Initial row count	Grows with	Estimated Monthly Growth	Initial Table Size	Table Size (6 months)	Comment
Dim_Time	1,826	Static	0	0.2 MB	0.2 MB	
Dim_Product	24	New products	0	0.2 MB	0.2 MB	
Dim_Policyholder	4,000	New policyholders	5%	250 MB	370 MB	
Dim_Broker	2,500	New brokers	1%	100 MB	106 MB	
Fact_Quote						
Consists of						
SourceSys1	2,143,322	For each quote	5%	5 GB	6.5 GB	Keep one year history
SourceSys2	2,323,430	For each quote	3%	3 GB	3.5 GB	Keep one year history
All Tables				8.354 GB	10.480 GB	
Fact table indexes	**Key Indexes**					
Dim_Time_idx	1			0.1 MB	0.1 MB	
Dim_Product_idx	1			0.1 MB	0.1 MB	
Dim_Policyholder_idx	1			50 MB	65 MB	
Dim_Broker_idx	1			10 MB	11 MB	
Fact_Quote_idx	4			1 GB	1.1 GB	
Total Indexes				1,06 GB	1.176 GB	
Temp space needed				2 GB	2.5 GB	
Total Space				**11.414 GB**	**14.156 GB**	

Figure 6-29 Database size and index plan

The database size and index plan reflects the following:

- Names of the tables in the data warehouse
- Estimated rows

- Potential growth
- Growth rate
- Initial size
- Estimated size over 6 months

Indexes and their estimates figure in the bottom half of the list. Partitioning information can also be included in the database size and index plan.

6.3.4.4. Physical Design – Develop aggregation plan

The business requirements that were gathered should highlight what needs to be aggregated. Based on this, a separate physical model similar to Figure 6-28 should be created with the necessary aggregate fields. It also should be sized and indexed, as was done for the atomic data table in Figure 6-29.

6.3.4.5. The Object Oriented Data Warehouse physical design

Owing to the nature of relational databases, OODW uses the same physical design as traditional Kimball DW development.

6.3.5. Data staging

Kimball *et al.* (1998:610) follows the under-mentioned ten-step overview when planning and implementing a data staging environment:

Plan:

1. Create one page source-to-target schematic flow
2. Test, choose and implement data staging tool
3. Create schematic plan to illustrate complex data restructuring and transformation and job sequencing

Dimension loads:

4. Build and test static dimension load
5. Build and test the slowly changing process
6. Build and test the remaining dimension

Fact table and automation:

7. Build and test historical fact table loads
8. Build and test the incremental load process
9. Build and test the aggregate table loads
10. Design, build and test the staging application automation"

In order to use the ten-step overview, the following main OO phases should be investigated:

- Requirements gathering
- Requirements analysis
- Design
- Implementation
- Testing

The requirements are gathered at the beginning of the project, followed by the analyses thereof. The latter is done during the dimensional modelling and technical architecture modelling stages.

The analysis of the backroom architecture serves as the planning part for the ten-step overview (steps 1 – 4). It provides the developers with the one page schematic flow, the strategy of the data stage tool to be implemented and a detail schematic plan of the data restructuring and transformation process.

In terms of OO phases, the dimension loads (step 4 – 6) and the fact table and automation (steps 7 – 10) consist of a design-, implementation-, and testing phase. Data staging in terms of OO phases will be discussed in the in the rest of section 6.3.5.

6.3.5.1. Data Staging – OO Design

The data staging environment has two major design areas:

- Dimension table loading

- Fact table loading and automation

The analysis documents available for the designs are:

- Dimension model designs
- Entity Relationship models of the source systems
- High level schematic plan
- Detail schematic plans
- Business rules

Based on the analysis documents listed above, the following need to be created for each dimensional load and fact table load in the data staging area:

- State chart model
- Entity relationship model
- Collaboration model between ETL processes

The state chart model should illustrate extraction of the data from its starting point (the source) to the transformation and from its conditions to the end point (the dimensions or fact tables). Figure 6-30 is an example of a state chart diagram for one dimension.

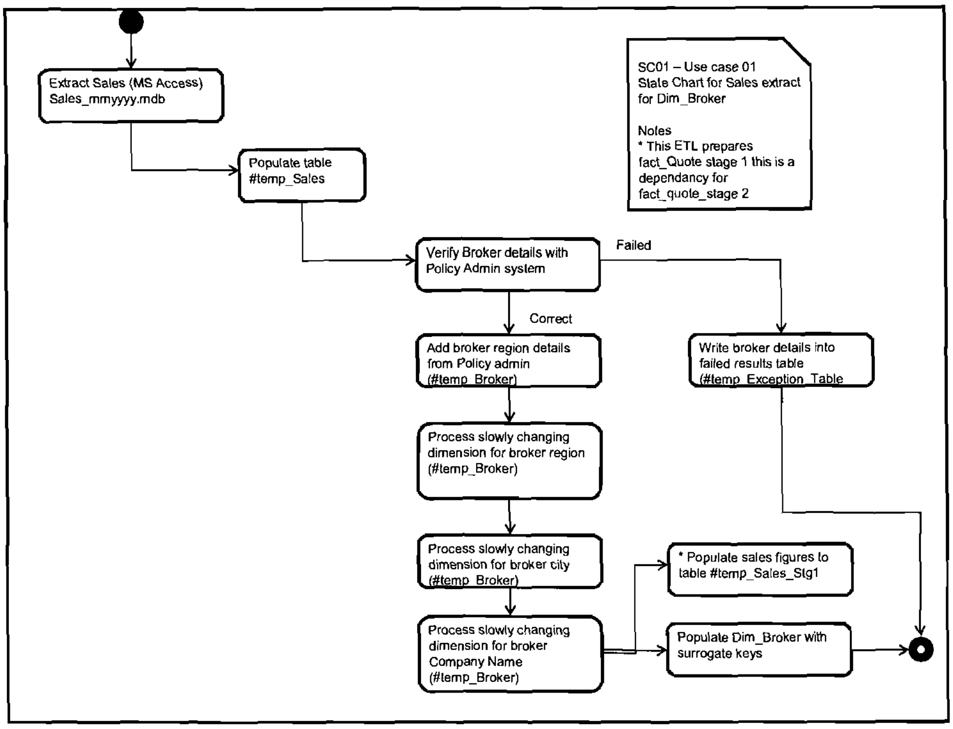


Figure 6-30 State chart model (SC01) extract for broker dimension (part of use case 01)

Figure 6-30 illustrates one ETL process that is derived from the detail schematic plan in Figure 6-26. The latter illustrates the source (Extract Sales (MS Access)) and different processes applied to the data up to the doughnut mark (bottom right hand corner). The "* populate sales figures to table #temp_Sales_stg1" process is a preparation for a next ETL process.

Accompanied with the state chart model, is an entity relationship model that illustrates the underling structure supporting the ETL. Figure 6-31 is an example of the ER model required to support the state chart model in Figure 6-30.

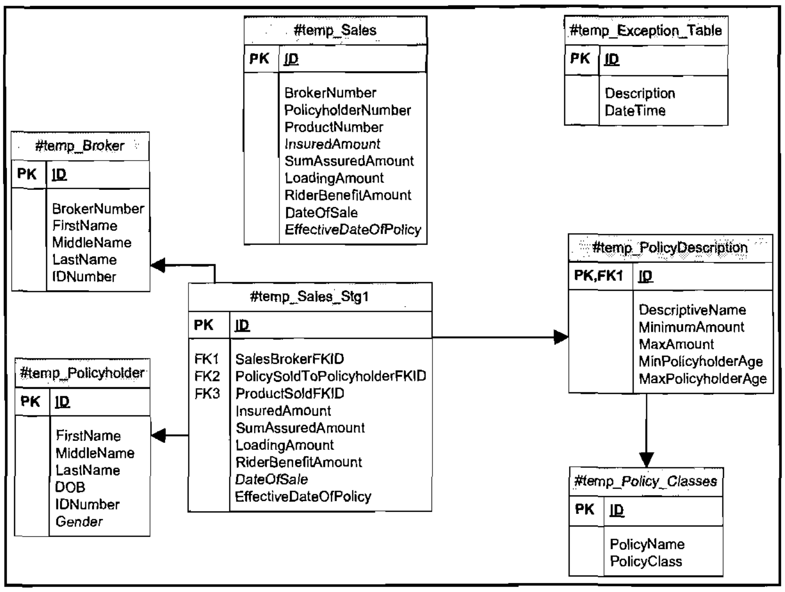


Figure 6-31 ER Model for staging environment for sales ETL

The reasons why an OODW uses a relational database system are the following:

- OO databases are not used commercially, while relational databases are mostly used.
- To create a flat file system is less appealing, as a file management system is needed for this. Relational databases come with very robust database management systems.
- A relational database uses industry standard query languages (SQL) and drives to access information, OO databases do not.

The *collaboration model is created once all state chart models and entity relationship* models are done for each dimension and fact table. The collaboration model provides a graphical representation of the interaction between the ETL processes and their dependencies. Figure 6-32 is an example of a collaboration diagram for a data warehouse.

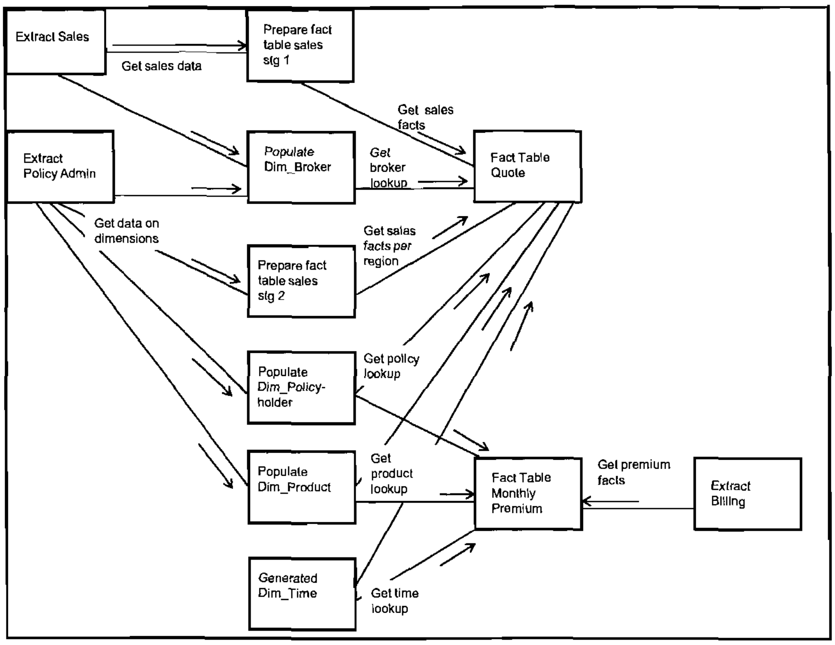


Figure 6-32 Collaboration diagram on the ETL for the data warehouse

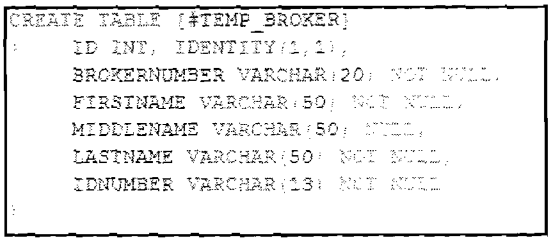
Figure 6-32 illustrates that the Sales fact table is dependent on the following tables:

- Temp_sales_stage_1
- Temp_sales_stage_2
- Dim_Broker
- Dim_Policyholder
- Dim_Product
- Dim_Time

The flow of data is from left to right therefore tables listed on the right hand side of the collaboration diagram is dependant on tables connected to its left. The above tables (except for Dim_time) in turn are dependent on the sales extract and the policy admin extract. The Monthly premium fact table is dependent on all of the listed tables (except for the sales fact tables) and the billing extract. This gives the developer a clear indication of the dependency and priority of each ETL job.

6.3.5.2. Data Staging – OO Implementation

The implementation of the dimension and fact table loads is supported by the state chart and ER model designs created. The ER model is implemented by using structured query language (SQL) scripts. Figure 6-33 is an example of a piece of SQL script that implements the #temp_broker table.



```
CREATE TABLE [#TEMP_BROKER]
(     ID INT, IDENTITY(1,1),
      BROKERNUMBER VARCHAR(20) NOT NULL,
      FIRSTNAME VARCHAR(50) NOT NULL,
      MIDDLENAME VARCHAR(50) NULL,
      LASTNAME VARCHAR(50) NOT NULL,
      IDNUMBER VARCHAR(13) NOT NULL
)
```

Figure 6-33 Example of SQL script to create the #temp_broker table

The state chart model and the ER model can be implemented by employing a data manipulation tool, such as data transformation services (DTS) in Microsoft SQL Server, or shell scripting in UNIX.

Figure 6-34 is a typical DTS screen shot used to populate a dimension or fact table. The DTS package is implemented according to the designs of Figure 6-30 (state chart model) and Figure 6-31 (ER model).

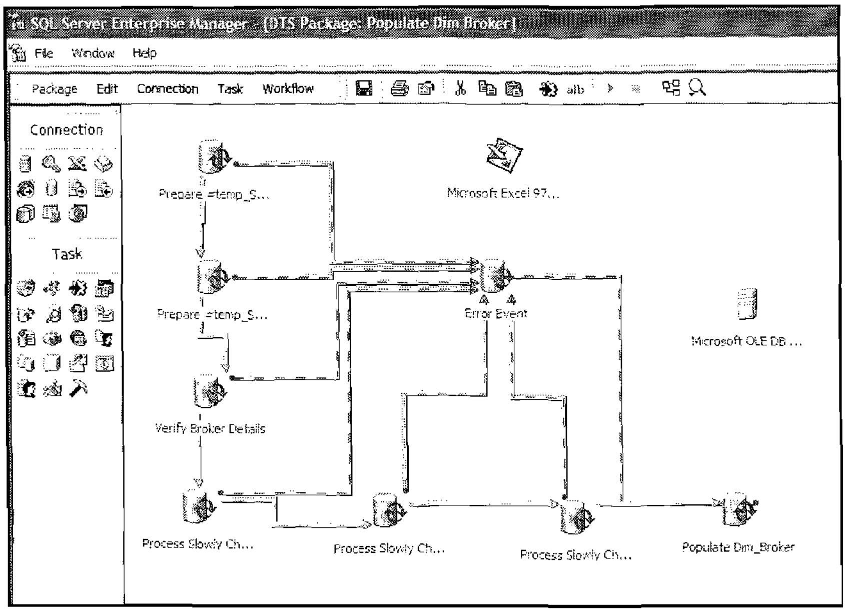


Figure 6-34 DTS Example for populating dimension broker

Kimball *et al.*(1998:617) recommends that the static dimensions should be implemented firstly, followed by the remaining dimensions and the fact table.

6.3.5.3. Data Staging – OO Testing

Kimball *et al.*(1998:631) suggests that audit statistics should be kept on all loads, thus allowing the following techniques to be applied for data quality assurance (Kimball *et al.* 1998:658):

- Cross footing – different queries are executed against the source system at different levels to compare the results
- Manual examination – consolidate data from different systems and investigate critical data points for acceptable ranges or exceptions
- Process validation – involves investigating the process flow of the data in the data warehouse

The testing can be done in phases with each load serving as a phase.

This section on DW data staging concludes with a discussion on the applicability of OO techniques to DW data staging.

6.3.5.4. The Data Warehouse data staging based on an OO approach

In terms of the data staging phase, the differences between the traditional Kimball *et al.* (1998) data warehouse development and the Object Oriented Data Warehouse (OODW) are the following:

- OODW splits the modelling of data staging into three phases, namely OO design, OO implementation and OO testing.
- OODW uses OO techniques such as state chart modelling and collaboration modelling to design the ETL loads.

Traditional Kimball DW development does not make use of the above in developing data warehouses. However, like in Kimball DW development, OODW also uses the following techniques:

- Dimensional models, ER models of the source systems, high level schematic plans, detail schematic plans and business rules.
- Testing, which in OODW is seen explicitly as a separate sub-phase, whereas in traditional Kimball DW, it forms an integrated part of the data staging development.

6.3.6. End use application

The end user application supports the front end query services of the data warehouse. These services are:

- Warehouse browsing
- Access and security
- Query management
- Reporting

The requirements of the end user applications are gathered within the business requirements definition phase. This determines whether a custom development or an off-the-shelf product is viable for implementation.

Not all requirements can be met with an off-the-shelf product. In such cases, a custom development is preferable. The development should follow the normal OO pattern, as it is regarded a project within its own right (Figure 6-1). These applications use the data warehouse as an input source to provide a service to the end user.

6.3.7. Deployment

According to the OO model defined in section 6.2. , the implementation phase involves the activities listed in Figure 6-35.

Phase	Activity
Object Oriented Implementation	• Code development • Component packaging • Deploy packages

Figure 6-35 Implementation model

The code development involves creating the scripts that support the data warehouse. These scripts are packaged into components. The components should correspond to the use cases, thus allowing the use case facilitating the quote business process, to contain all the code involved in implementing the latter.

The deployment of these packages should be implemented by using different environments. For example, the development should first be done on a development platform and on completion, promoted to a quality assurance (QA) environment for thorough testing. Once the package has passed all necessary tests, it can be promoted to the production data warehouse.

6.3.8. Maintenance and growth

Kimball *et al.* (1998:718) stresses that the data warehouse should serve the needs of the business users for it to be successful. This statement implies that the data warehouse lifecycle should follow a spiral approach to its development. OO development also follows an iterative lifecycle approach. Figure 6-36 is an illustration suggesting the life cycle of developing a DW using OO techniques.

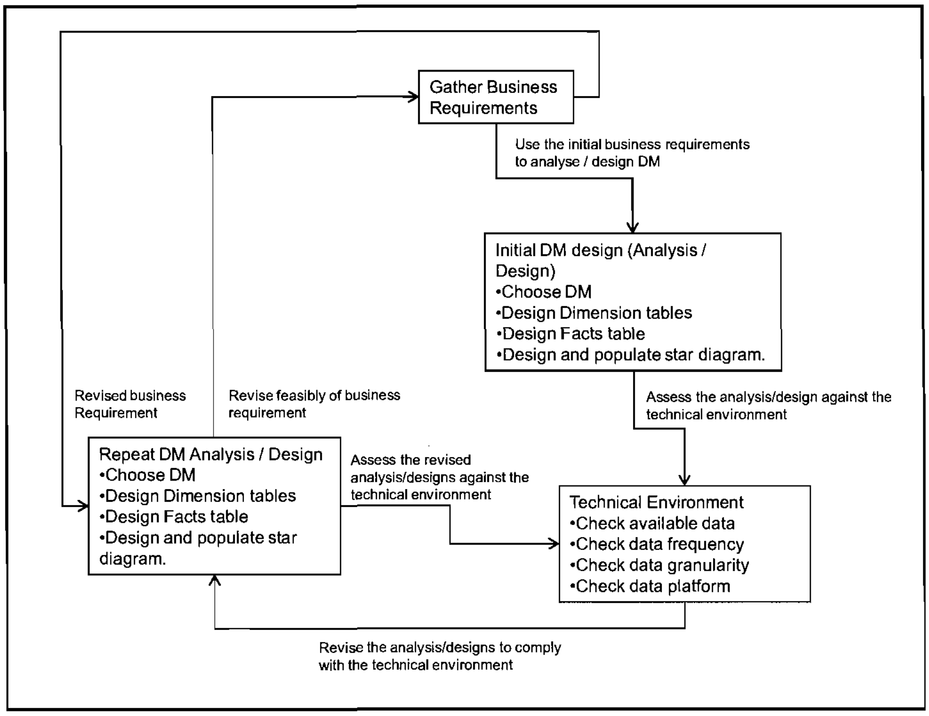

Gather Business Requirements
Use the initial business requirements to analyse / design DM
Initial DM design (Analysis / Design)
•Choose DM
•Design Dimension tables
•Design Facts table
•Design and populate star diagram.
Revised business Requirement
Revise feasibly of business requirement
Assess the analysis/design against the technical environment
Repeat DM Analysis / Design
•Choose DM
•Design Dimension tables
•Design Facts table
•Design and populate star diagram.
Assess the revised analysis/designs against the technical environment
Technical Environment
•Check available data
•Check data frequency
•Check data granularity
•Check data platform
Revise the analysis/designs to comply with the technical environment

Figure 6-36 Lifecycle of a data mart development

Figure 6-36 suggests that the lifecycle start with the requirements gathering phase. Thereafter the analysis is done. Based on the assessment of the technical environment the design is done. This step is not a once off step and is iteratively done until the design is compatible with the technical environment. In some cases if the design is not technically feasible one should reconsider some of the business requirements.

This section concludes with a discussion on the business dimensional lifecycle approach and the applicability of OO techniques to the methodology in question.

6.3.9. Summary: The business dimensional lifecycle approach based on an OO approach

The business dimensional lifecycle approach is a top down approach, starting with the requirements and working towards a solution. This concept works well with the OO model defined, as this model follows a similar lifecycle.

The techniques of the requirements phase of the OO model can as such be implemented in the business requirements definition phase of the business dimensional lifecycle. The OO techniques working with the business requirements definition are:

- Essential use case diagram
- Essential use case model
- User interface prototyping
- Domain modelling
- Supplementary documentation

The concepts of an essential use case diagram and model can be altered to accommodate two levels of analysis. The first type of use case can be seen as a data warehouse use case containing the inputs, departments or units of business and users of the business. Each of the departments or units of business contains one or multiple processes of business documented in the process use case diagram and process use case model.

The OO techniques used in the dimensional modelling phase of the Business dimensional lifecycle are more discrete. The dimensional modelling phase can be used during the analysis and design phases. The following OO techniques are recommended for the analysis phase of dimensional modelling:

- System use case
- Sequence diagram

The system use cases can also be used to create a DW system use case and a business system use case. The DW system use case contains the technical specification for the inputs, outputs and the grain of the use case. The business system use case contains high-level implementation decisions, such as the screen numbers and properties needed in the use case.

From here onwards, the following analysis techniques in the business dimensional lifecycle are used:

- Data warehouse bus architecture matrix (derived from the DW use case list and business process use case list)
- Fact table diagram
- Dimension table detail
- Identify source data

Based on the above analysis techniques, the dimensional tables are developed.

The technical architecture model can be implemented in an analysis and design fashion. The technical architecture model is divided into two parts, i.e. the back room architecture and the front room architecture.

The back room architecture of the technical architecture model involves the analysis of the following:

- Source systems
- Data staging services

Based on the analysis, the backroom designs can be done. Most of these designs are done during the physical design phase of the business dimensional lifecycle.

The analysis of the front room architecture of the technical architecture model is based on the business requirements definition. Depending on the requirement of the business, the front room architecture application can involve a full OO development lifecycle. The same argument is valid for the design of the front room architecture applications.

The physical design phase in the business dimensional lifecycle approach is implemented by using a design, implementation and testing phase. The design involves the design of the data warehouse, as well as the design of the data staging area.

The design of the data warehouse entails the following:

- Defining standards
- Designing physical tables and columns
- Estimating database size and index plan
- Designing aggregates

The design of the data staging area entails the following:

- Designing a state chart diagram based on the analysis of the data staging services in the technical architecture
- Designing an ER model that supports the state chart model
- Designing a collaboration model that depicts the overall ETL schema

The implementation of the data warehouse and the data staging involves coding of the designs created.

The testing of the data staging phase follows the same approach than the OO approach. The testing of ETL jobs are done in phases suggesting that it is unit tested. Audit statistics is used to test the data staging.

The end user application supports the front end of the technical architecture design. As previously indicated, a complete OO model can be applied to the development of these applications.

Deployment of the Business dimensional lifecycle approach follows a similar approach than the OO model. The implementation is done in phases and not in total.

6.4. Data warehouse development using the d ata-driven methodology phases

The objective of this discussion is to describe how a data warehouse should be built if the approach of Inmon (1996) is implemented in an object-oriented manner.

The data-driven methodology (Inmon, 1996) illustrated in Figure 6-37 is discussed in the following section.

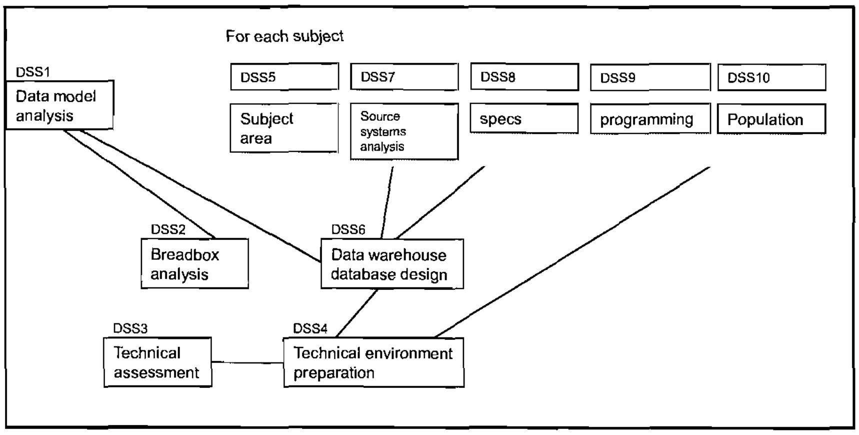


Figure 6-37 Data driven methodology (Inmon, 1996:344)

Unlike the business dimensional lifecycle approach (Kimball *et al.*, 1998) which follows a requirements-driven approach, the data-driven methodology does not start with the business requirements for a data warehouse, but with the data from the operational systems.

The lifecycle of OO development follows the following phases sequentially and iteratively:

Requirements → Analysing requirements → Designing solution → Implementing solution → Testing solution.

In terms of OO phases, the data-driven approach can be assumed to follow the phases in the following sequence:

Analysis (of the environment) → Technical requirements gathering* → Designing solution → Implementing solution → Testing solution.

Technical requirements gathering is regarded as the requirements needed to build the data warehouse and not the requirements needed from the end user.

The rest of the discussion will elaborate on the sequence of the data-driven methodology.

6.4.1. Data model analysis

Inmon (1996:81) argues that there are three types of data models applying to the architectural environment:

- Corporate data model – the model containing the primitive data elements used within the corporate.
- Operational data model – the model based on the corporate data model with operational data included.
- Data warehouse model – the model based on the corporate data model with added elements, such as:
  - Time
  - Derived data where needed
  - Artefacts for relationships

The data-driven methodology assumes the existence of a corporate model and an operational data model on which the data warehouse can be developed.

Inmon (1996:85) also argues that a data model has three levels:

- High-level model – the entity relational model (ER) highlighting the entities and the relationship between these entities in the corporation.
- Mid-level model – the data item set model (DIS) detailing the major subject areas.
- Low-level model – the physical model, based on the DIS model and containing relational tables and keys.

The rest of the discussion will be based on the corporate ER model illustrated in Figure 6-38.

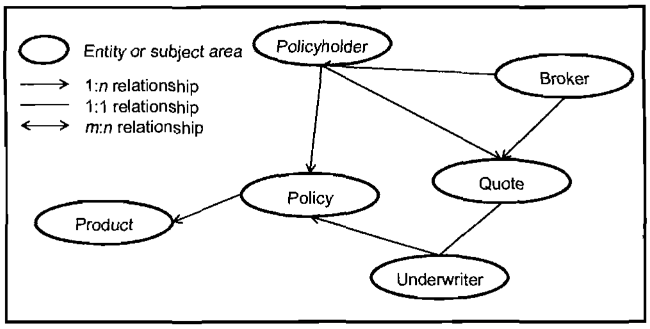


Figure 6-38 Corporate entity relationship diagram for insurance company example

The above ER model can be transformed into a UML class diagram. Figure 6-39 is an example of such a transformed class model.

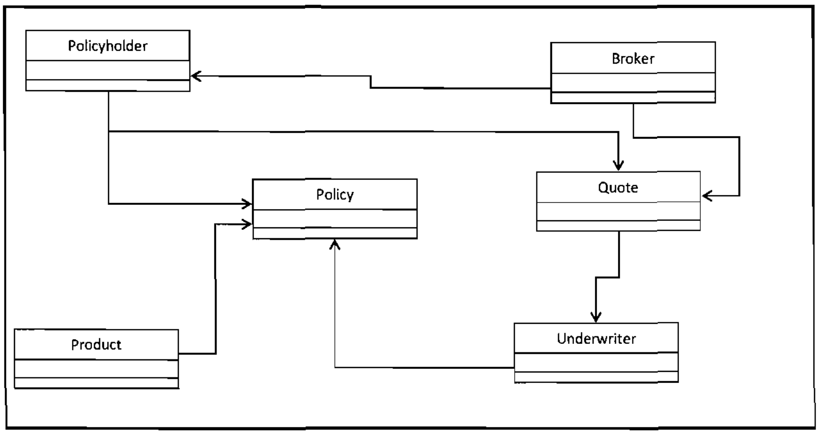


Figure 6-39 UML class diagram

From Figure 6-38, the following subject areas or entities can be identified:

- Policyholder
- Broker
- Quote
- Underwriter
- Policy
- Product

A subject area should be treated as a use case, and therefore a list of possible use cases should be created (refer Table 6-2 Example of a list of essential use cases generated from the use case diagram). The description of the use case should provide more detailed information on what the subject area entails. Each subject area has a DIS defined. Figure 6-40 is an example of a DIS for the product entity.

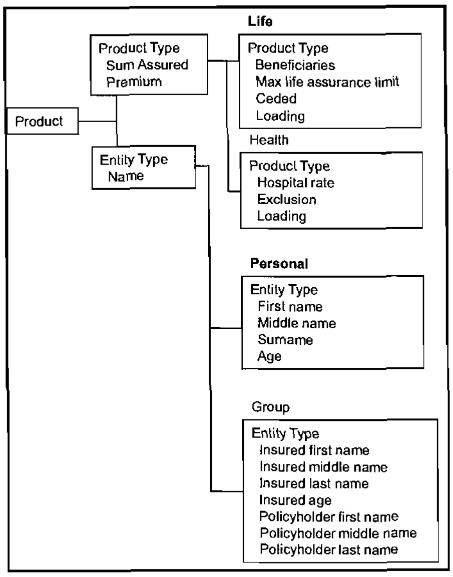


Figure 6-40 Corporate data item set for product

The product entity in Figure 6-40 serves as the key. The primary grouping of the product entity is the product type and the secondary grouping is the entity type. The product type can be a life product or a health product. The entity type can be a personal type or group type. For the purpose of this discussion, the DIS for all the entities or subject areas will not be expanded.

The physical model is based on the DIS of each entity of a subject area. Figure 6-41 is an example of the physical data model for the product DIS.

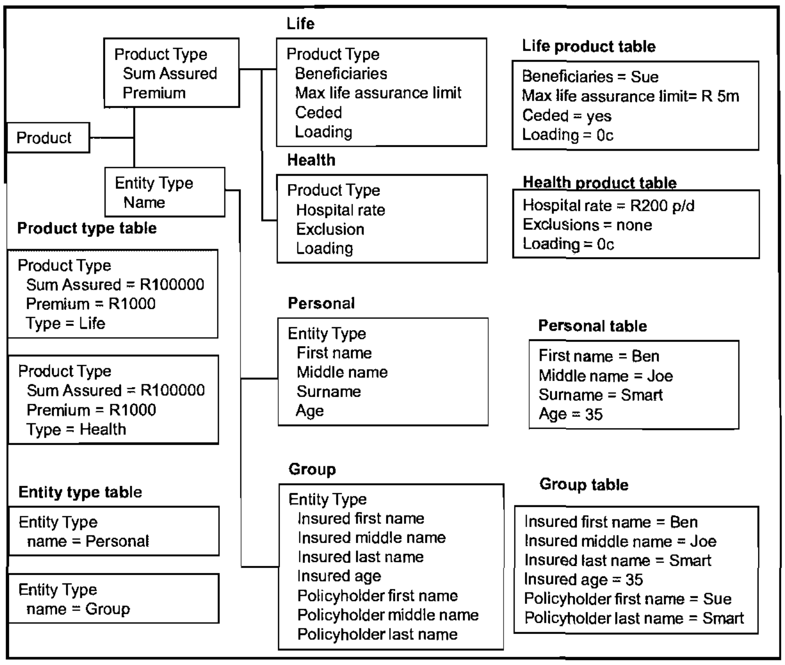


Figure 6-41 Physical data model for product DIS

The product type table contains an example of the two types of products, i.e. life and health. The entity type table reflects entries for both personal and group types of cover. Each of the table types (life, health, personal and group) contains sample data.

The purpose of the data model analysis is to identify the major subject areas. For each subject area, the following should be identified (Inmon, 1996:335):

- Sub-types
- Attributes
- Groupings of data
- Keys

Based on the corporate ER model shown in Figure 6-38, the subject areas can be defined as quote and policy, as these entities contain the information of a process. The function of the DIS model is to define the sub-types, groupings, attributes and keys.

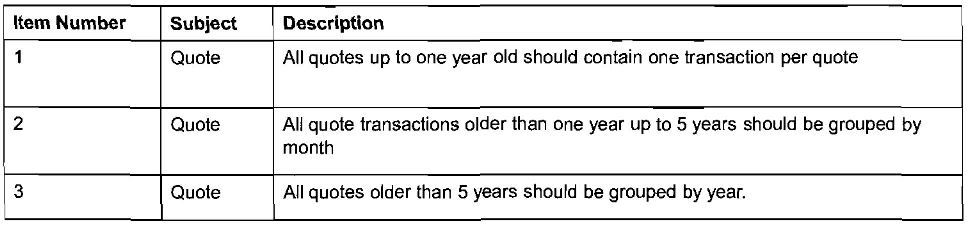
For the purpose of data model analysis, it is not possible to apply an OO technique, but the analysis can be categorised to become part of an OO analysis phase.

6.4.1.1. Breadbox analysis

The function of the breadbox analysis is to estimate the required level of granularity of data in the data warehouse (Inmon, 1996:336).

A document (database index and sizing), similar to the one illustrated in Figure 6-29, should be created. The main difference between the data-driven approach and the business dimensional lifecycle approach is that tables listed in the document will not contain tables for the data warehouse, but tables listed in the corporate data model.

This analysis should produce a document specifying that grain of the data. Figure 6-42 is an example of such a document.



Item Number	Subject	Description
1	Quote	All quotes up to one year old should contain one transaction per quote
2	Quote	All quote transactions older than one year up to 5 years should be grouped by month
3	Quote	All quotes older than 5 years should be grouped by year.

Figure 6-42 Document containing the different grains needed for the subject area

6.4.1.2. Technical assessment

The technical assessment involves investigating the requirements for managing the following (Inmon, 1996:337):

- Large volumes of data
- Access to data
- Sending and receiving of data to a wide variety of technologies
- Data loading and manipulation
- Access to a set of data

Managing large volumes of data – OO Analysis

The database index and sizing document (Figure 6-29), in conjunction with the breadbox analysis document (Figure 6-42) should facilitate an estimation of the data volumes of the different subjects. The main difference between the database index and sizing document specified in the business dimensional lifecycle, and the data-driven methodology, is that specific table names to be used in the data warehouse will not be detailed. Therefore, only the subject areas specified in the data model should be used.

Managing access to data – OO Analysis

Managing access to data involves specifying the security level of the different subjects. For example, it is not advisable for brokers to access policyholders listed under other brokers. A document specifying users or groups allowed to access specific data should be created.

Sending and receiving data to a wide variety of technologies – Analysis

This analysis involves specifying the current layout of the organisation's environment, as well as the flow of data and technologies used. Figure 6-43 is a diagram of the layout and data flow for the insurance company.

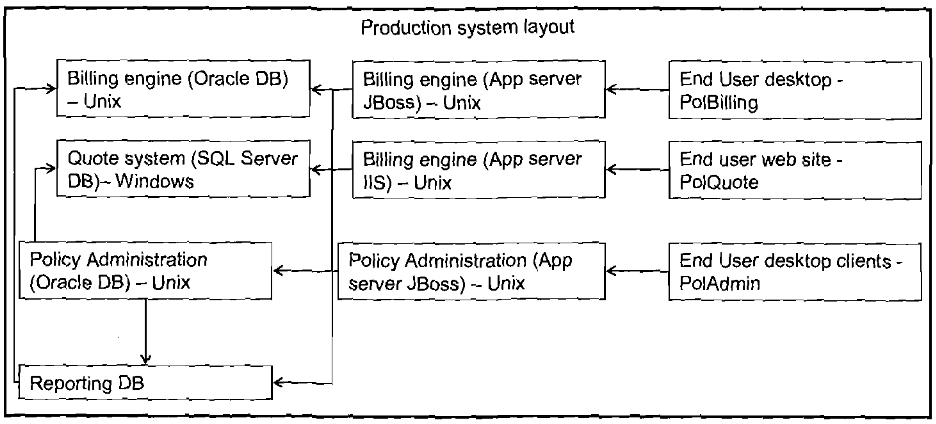

Production system layout
Billing engine (Oracle DB) – Unix
Billing engine (App server JBoss) – Unix
End User desktop - PolBilling
Quote system (SQL Server DB)- Windows
Billing engine (App server IIS) – Unix
End user web site - PolQuote
Policy Administration (Oracle DB) – Unix
Policy Administration (App server JBoss) – Unix
End User desktop clients - PolAdmin
Reporting DB

Figure 6-43 Example of the production system layout

In Figure 6-43, the data flow specifies three production systems i.e. the billing engine, the quoting system and the policy administration. All end user applications flow towards these systems. Both the billing and quote system use information entailed in the policy administration system.

Manage data loading and manipulation – OO Analysis

This analysis involves specifying the source systems and the extract transformation, as well as loading of the source data to the data warehouse. For this process, a list of source systems similar to Figure 6-19 should be defined. Furthermore, a high level schematic plan similar to Figure 6-23 and based on the source systems and the defined subject should be created.

Access a set of data – OO Analysis

This analysis involves specifying the types of tools used to access (interface) the data warehouse. For this analysis, a requirements definition should be created, specifying the types of users and the tools used to access the data. Examples of these are the following:

- Typical report writer – SQL tool to access data
- Manager has a set of reports to access – these reports use the same query to access data

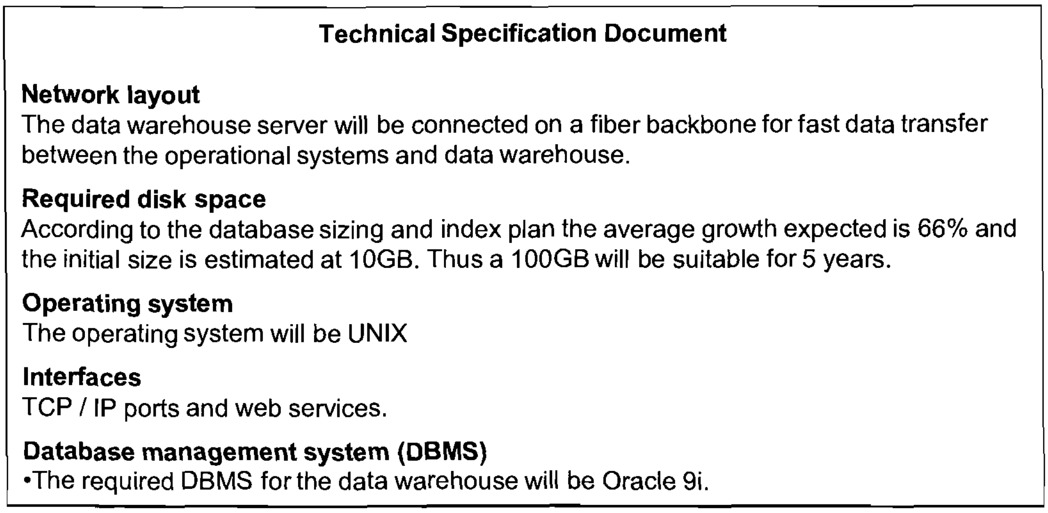
The same procedure discussed in section 6.4.1 should be followed to create the analysis documents for data access.

6.4.1.3. Technical environment preparation

The technical assessment serves as the technical design hosting the data warehouse. It focuses on the following (Inmon, 1996:338):

- Network
- Amount of disk space required
- Operating system
- The interfaces specified for the data warehouse
- Software managing the data warehouse

Based on the analysis documents gathered during the technical assessment, a document containing the technical specifications should be created. The technical specification document should then be implemented. Figure 6-44 is an example of a technical specification document.



Technical Specification Document

Network layout
The data warehouse server will be connected on a fiber backbone for fast data transfer between the operational systems and data warehouse.

Required disk space
According to the database sizing and index plan the average growth expected is 66% and the initial size is estimated at 10GB. Thus a 100GB will be suitable for 5 years.

Operating system
The operating system will be UNIX

Interfaces
TCP / IP ports and web services.

Database management system (DBMS)
•The required DBMS for the data warehouse will be Oracle 9i.

Figure 6-44 example of a technical specification document

6.4.1.4. Subject area

For each subject area, the following must be done:

- Source system analysis
- Data warehouse design
- Program specification
- Population

Inmon (1996:339) explains that the subject area selected first, should be small enough to allow easy changes and large enough to be meaningful. This statement implies that the processes applied to each subject area will be iterative, while the result of the population process will serve as a testing environment.

Up to this stage, no specific use case was focused on, as the use case should be chosen during the subject area phase. For the purpose of this discussion, the quote subject area will be used as in the business dimensional lifecycle approach.

6.4.1.5. Source system analysis

The source system analysis identifies the source data that will be used for the star join. A source data list identifying all the source data should be created. This list should look similar to the source data list in Figure 6-27. The ER model of the source data should be investigated to identify the tables and attributes that form part of the subject area in question (Figure 6-31).

Once the source data is identified, the design of the star join for the subject can be created, and this design can then be integrated into the data warehouse design. This will be discussed in section 6.4.1.6.

6.4.1.6. Data warehouse database design

The data warehouse database design is subject to the design of the subject areas. It therefore should not be treated as a sequential phase in the development of the subject areas, but rather be designed iteratively as the design of the subject area progresses.

The analysis documents required for the design of the data warehouse are the following (Inmon, 1996:339):

- Breadbox analysis
- Source system analysis
- Data model analysis

Inmon explains that the design of the data warehouse should be based on the corporate data model (Inmon, 1996:81) and should have the following characteristics (Inmon, 1996:339):

- Accommodate different levels of granularity

- Subject-oriented
- Contain only primitive and derived data and no operational data
- Time variance on every record

Based on the subject area in question, the star joins should be created. A star join, as discussed in chapter 4, is the same as a dimensional diagram, both these concepts making use of dimensional tables (the tables describing the fact) linking to a fact table (the table containing all the numeric facts). See Figure 6-21 for an example of a star join or dimensional model.

Based on the logical design of the star join, the physical design is done. A physical data model is then used (as with the business dimensional lifecycle) to implement the star joins (Figure 6-28).

The design of the star joins for each subject area in the data warehouse should be the same grain, as these star joins should be compatible and fit into the data warehouse architecture. For example, the level of grain of the product subject area is per product. If the quote subject area is per quote (which contains multiple products), it cannot be compared on an one to one basis and therefore requires a finer level for such a comparison.

6.4.1.7. Specification

The specifications phase serves as the analysis and design of the data warehouse ETL process (Inmon, 1996:342).

Specification – OO Analysis

The section on managing data loading and manipulation discussed the analysis of managing the loading of data and also referred to the high-level schematic plan. The specification analysis details the high-level schematic plan (Figure 6-26) into a detailed

6.4.1.9. Population

Inmon defines the population phase as "nothing more than the execution of the decision support system programs previously developed" (Inmon, 1996:343). This step produces a fully functional data warehouse.

Population – OO Testing

Although population is seen as the last phase, it should be considered as the testing ground. A series of steps is followed to develop a subject area in the data warehouse. Once the subject area is developed, it should be tested. Testing a specific subject area at a time ensures unit testing. Unit testing on a subject area can be accomplished by using audit statistics (section 6.3.5.3.) on the star joins.

6.5. Summary

This chapter deals with the development of data warehouses using OO concepts, tools and techniques. The phases of the b usiness dimensional lifecycle approach and the data-driven methodology are used in an OO model derived from chapter 4. For the purpose of this summary, the derived model will be referred to as the OO model.

The Business dimensional lifecycle approach is a top down approach, starting with the requirements and working towards a solution. This concept works well with the OO model defined, as the model follows a similar lifecycle.

The techniques of the requirement phase of the OO model can as such be implemented in the business requirements definition phase of the Business dimensional lifecycle. The OO techniques working with the business requirements definition are:

- Essential use case diagram
- Essential use case model
- User interface prototyping
- Domain modelling

- Supplementary documentation

The concepts of an essential use case diagram and model can be altered to accommodate two levels of analysis. The first type of use case can be seen as a data warehouse use case containing the inputs, departments or units of business and the users of the business.

Each of the departments or units of business contains one or multiple processes of business documented in the process use case diagram and process use case model.

The OO techniques used in the dimensional modelling phase of the Business dimensional lifecycle are more discrete. The dimensional modelling phase can be used during the analysis and design phases. The following OO techniques are recommended in the analysis phase of dimensional modelling:

- System use case
- Sequence diagram

The system use cases can also be used to create a DW system use case and a business system use case. The DW system use case contains the technical specification for the inputs, outputs and the grain for the use case. The business system use case contains high-level implementation decisions, such as the screen numbers and properties needed in the use case.

From here on, the following analysis techniques in the Business dimensional lifecycle are used:

- Data warehouse bus architecture matrix (derived from the DW use case list and business process use case list)
- Fact table diagram
- Dimension table detail

plan for the subject area in question. The analysis of the specification also includes business rules applying to, or affecting the transformation of data.

Specification – OO Design

The analysis documents available for the specification design are:

- Star joins
- ER models of the sources systems
- High level schematic plan
- Detail schematic plans
- Business rules

Based on the above analysis documents, the specification design for the subject area in question can be created. The design includes the following documents:

- State chart model for the extract to star joins (Figure 6-30)
- ER model to support the state chart model (Figure 6-31)

Although the specification design focuses only on one subject area at a time, the possibility of reusing dimensions should be investigated. If dimensions can be reused, a collaboration model should be created to depict the dependencies of the star joins and prioritise the ETL (Figure 6-32).

6.4.1.8. Programming

The programming of the specifications mainly involves utilising the technology specified to implement the designs for the subject area in question. The ER model created for the state chart model can be implemented by using SQL scripts (Figure 6-33). The state chart model can be implemented by using DTS packages (Figure 6-34).

The collaboration model (Figure 6-32) serves as a guide to know which ETL jobs to implement and when it should be done.

- Identify source data

Based on the above analysis techniques, the dimensional tables are developed.

The technical architecture model can be implemented in an analysis and design fashion. The technical architecture model is divided into two parts, i.e. the back room architecture and the front room architecture.

The back room architecture of the technical architecture model involves the analysis of the following:

- Source systems
- Data staging services

Based on the analysis, the backroom designs can be done, as most of these designs are done during the physical design phase of the Business dimensional lifecycle.

The analysis of the front room architecture of the technical architecture model is based on the business requirements definition. Depending on the requirements of the business, the front room architecture application can involve a complete OO development lifecycle. The same argument is valid for the design of the front room architecture applications.

The physical design phase in the Business dimensional lifecycle approach is implemented using a design, implementation and testing phase. The design involves the design of the data warehouse, as well as the design of the data staging area.

The design of the data warehouse entails the following:

- Defining standards
- Designing physical tables and columns

- Estimating database size and index plan
- Designing aggregates

The design of the data staging area entails the following:

- Designing a state chart diagram based on the analysis of the data staging services in the technical architecture.
- Designing an ER model supporting the state chart model.
- Designing a collaboration model depicting the overall ETL schema.

The implementation of the data warehouse and the data staging involves the coding of the designs created.

The testing of the data staging phase uses the same approach than the OO approach. The testing of ETL jobs are done in phases suggesting that it is unit tested. Audit statistics is a technique used to test the data staging.

The end user application supports the front end of the technical architecture design, and a complete OO model can be applied to the development of these applications.

The deployment phase in the Business dimensional lifecycle approach is similar to that of the OO model. The implementation is done in phases and not in total.

The second methodology to be discussed is the Data-driven methodology. This methodology is less compatible with the OO model. The main reason for this being that both the OO model and the Business dimensional lifecycle approach start with and depend on the requirements defined. The Data-driven methodology on the other hand does not start with the requirements, but rather focuses on the data available for the data warehouse as starting point.

The following phases in an OO approach can not be used:

- Data model analysis
- Breadbox analysis
- Technical assessment

The above phases focus on the data warehouse design as a whole. Based on these analysis documents, the technical environment is prepared.

The data model analysis can be used to create a list of use cases to which the subject areas correspond.

There are certain phases in the Data-driven methodology following the same approach than the OO model. These are:

- Subject area
- Source system analysis
- Data warehouse design
- Specification analysis
- Programming
- Population

The subject area involves selecting the subject area to be developed. A subject area can also be regarded as one use case.

The source system analysis phase is a phase identifying the sources needed for the subject area. This phase does not use any of the techniques in the defined OO model.

The design of the data warehouse follows an OO analysis and design phase. It does not use the OO analysis and design techniques, but rather techniques specific to databases and data warehouses, i.e. ER models and star joins (dimensional models).

The specifications phase is a design phase specifying how the implementation should be done. This phase follows a design phase in the defined OO model and uses the state chart and collaboration techniques.

The programming phase involves implementing the designs for the subject area.

The population phase is the product of all the phases in the Data-driven methodology, the outcome of which should be used as the testing platform. The testing can follow a unit testing approach as per OO testing.

Chapter 7 - Object-oriented implementation of a data warehouse

7.1. Introduction

In the first part of this chapter the research design for the study is discussed, followed by the study's action taking in the form of a DW prototype. A short discussion on the nature of the study follows as this has an impact on the research design. The research design involves choosing the research method and environment for implementing the research method. The detail of how the action plan is implemented is covered. The action plan is in the form of a DW prototype. A report on the research evaluation follows in the next chapter.

7.2. Research question and scope of study

The aim of the study is to develop a data warehouse using object-oriented techniques in a data warehouse development methodology. Thus the research question is:
Can a data warehouse be developed using a data warehouse methodology and incorporating object-oriented techniques?

To answer the above question, the researcher needs to implement the business development lifecycle approach using object-oriented techniques (discussed in chapter 6). The data-driven methodology can be used for further studies and will not be implemented for this research.

7.3. Nature of the study

The study focuses on the data warehouse development methodologies with the aim of incorporating different techniques from object-oriented development. Chapter 6 discussed methods to develop data warehouses using object-oriented concepts, tools and techniques where possible. Two approaches were covered, i.e. The Business

Development Lifecycle Approach (Kimball *et al.*, 1998) and the Data Driven approach (Inmon, 1996).

Very little literature on the OO development of DW is available. The work presented in chapter 6 will be used as a guideline for the development of the DW. OO methods will be explored in every phase of the systems development lifecycle.

7.4. Research method

Based on the nature of the study discussed in the previous section, one can use action research as a research method. Baskerville (1999:11) explains that the ideal situations for action research are:

- Research environment where the researcher is actively involved, with the expectation that both the researcher and the organisational benefits.
- Research environment were the knowledge obtained can be applied immediately.
- Research environment where research is a process of linking theory and practice.

The research environment for this study satisfies all of the above situations with the following reasons.

- The researcher was actively involved in the development lifecycle of the data warehouse and both the researcher and the organisation's expectations were set during the requirements meetings.
- The environment was favourable due to the OO development culture in the organisation and the need for a data warehouse.
- The need for a OO DW requires a study on OO DW as it is not commonly done in practice.

7.5. Research design

The discussion on the research design is according to the action research cycle in Figure 7-1. The theory on action research was discussed in chapter 2.

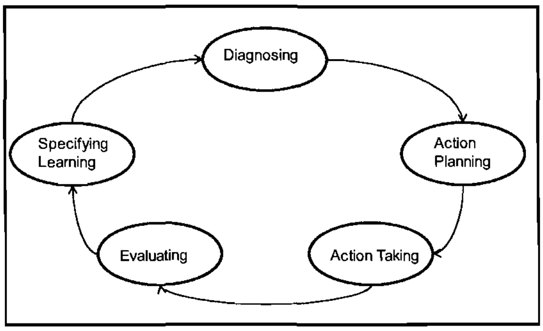


Figure 7-1 The Action Research Cycle (Baskerville, 1999:14)

Diagnosing – is the process of identifying the primary reasons why change is needed. The research question of this study aims to discover the benefits of using OODW to DW. OO has already proven to be very successful in organisations, although DW systems are traditionally not based on OO. The diagnosing environment for this research is discussed in section 7.6.1.

Action Planning – involves the researchers and practitioners to collaborate and produce actions that should relieve or improve the problems identified. A plan containing the necessary actions is created and carried out by means of a theoretical framework. The plan should establish the target and approach for change.

The study serves as the action plan. It starts with a literature study of what a methodology is and how it can be classified. It then studies common object-oriented methodologies, as well as common data warehouse methodologies. Based on the literature studies of object-oriented and data warehouse methodologies, a theory is created. This theory was discussed in chapter 6 and covered data warehouse development and the object-oriented methodologies. Therefore, the discussion in chapter 6, the data warehouse and object-oriented approach, serves as the action plan.

Action Taking – implements the action plan. The action is the proposed methodologies discussed in chapter 6 (Data warehouse and OO approach). The method of how a data warehouse can be created using the business lifecycle approach in an OO environment will be implemented. This is discussed in detail in section 7.6.

Evaluating – the outcomes of the action plan are evaluated. The evaluation determines whether the theoretical effects were realised and whether the problems identified are relieved, or not. If the changes implemented were successful, it must be determined whether the changes are the sole cause of the success. If the changes implemented were unsuccessful, a framework for a next iteration should be established. The detailed discussion on the evaluation of the study follows in chapter 8. The evaluation for the DW implemented is based on feedback from areas in business, such as service desks and managers of the affected departments.

Specifying learning – is the knowledge gained from the research. This is detailed in chapter 8.

7.6. Implementing the business lifecycle approach in an object-oriented fashion.

This section discusses the process that was followed to implement a data warehouse using one of the proposed methods described in the study. The development methodology used to create the data warehouse was described in section 6.3 (Data warehouse development using the Business dimensional lifecycle approach phases).

7.6.1. Diagnosis and Background to the data warehouse prototype implemented

The development of the data warehouse (DW) was done for a leading insurance company based in Johannesburg, South Africa.

The company uses several operational systems for its daily operations and had a need for a consolidated source of information. The main operational systems are mostly in-house developed systems using object-oriented (OO) methodologies and technologies.

The company has in excess of 200 000 policyholders and more than 600 000 policies worth of data. For certain insurance products, such as the Income Protector policy, transitional data is kept, thereby increasing the volume of data tremendously. The production systems total around 5 TB of data.

The company in need of a data warehouse and geared towards OO development, lends itself to being an ideal candidate for implementing the theory created in chapter 6 (Data warehouse and the object-oriented approach). The DW is implemented to be used as a reporting platform. This allows report writing, easy access to data and serves as a medium for running monthly reports, which previously had a negative impact on production systems.

The DW was developed by the researcher, a business analyst, the manager from the application support department and various key managers (the users) from different departments.

Project duration for the core analysis and development was about 6 months, while further development is planned for the DW.

The requirements for overall functionality of the DW were done by the manager of application support. The specific requirements for the data marts and reports were prepared through interviews with relevant parties and by the business analyst and researcher.

7.6.2. Business requirements definition

As described in section 6.3.1, the business requirements definition determines the following:

- Which data should be available in the DW?
- How this data should be organised?
- How often it should be refreshed?

The OO techniques described in the same section are illustrated in Figure 7-2.

Phase		Activity
Requirements	Gathering Requirements	• Essential Use Case Diagram • Essential Use Case • User Interface Prototyping • Domain Modelling • Supplementary Documentation
	Validating Requirements	• Use Case Scenario Testing • User Interface Walkthrough • Requirements Review

Figure 7-2 Requirements Model

The first activity was to create the essential use case diagram.

7.6.2.1. Essential use case diagram

The essential use case diagram is an activity discussed in chapter 4. Its function is to illustrate the interaction between actors and concepts in the problem domain. For this case study, the problem domain needed to be established. An in-depth study was done about the company's structure and the interaction between departments. The structure of the company is illustrated in Figure 7-3.

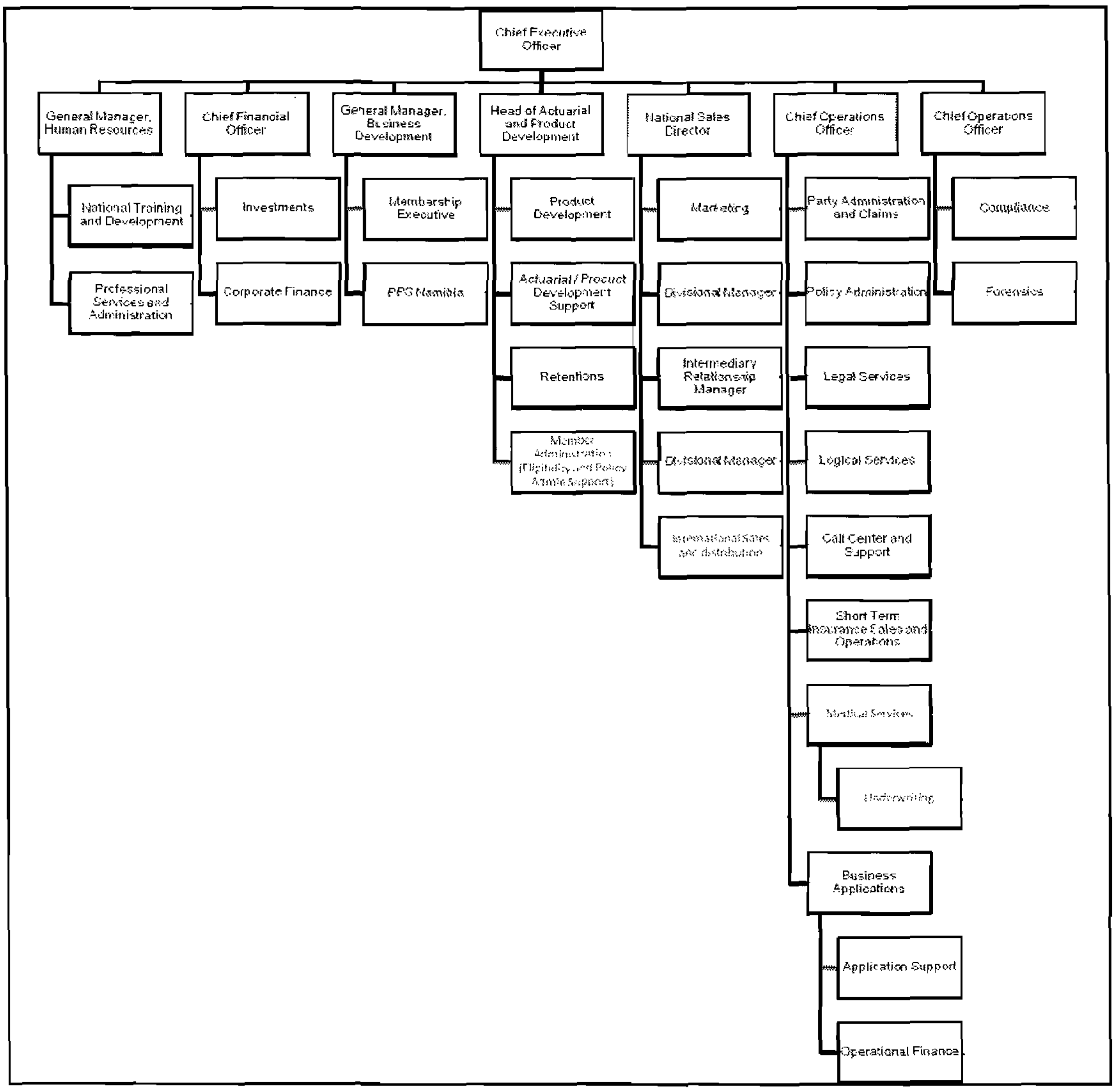


Figure 7-3 Insurance company organisational structure (human resources department)

Figure 7-3 was supplied by the human resources department. This organisational structure served as the basis for creating the essential use case diagrams. Section 6.3.1.1 introduces the concept of splitting the essential use case diagram into a data warehouse use case diagram and a business process use case diagram. The concept of the data warehouse use case diagram is to illustrate the "big picture".

Data warehouse use case diagram

Figure 7-4 shows the data warehouse use case diagram created. This was done with the help of the organisational structure (Figure 7-3) and by facilitated sessions with a business analyst in the application support department.

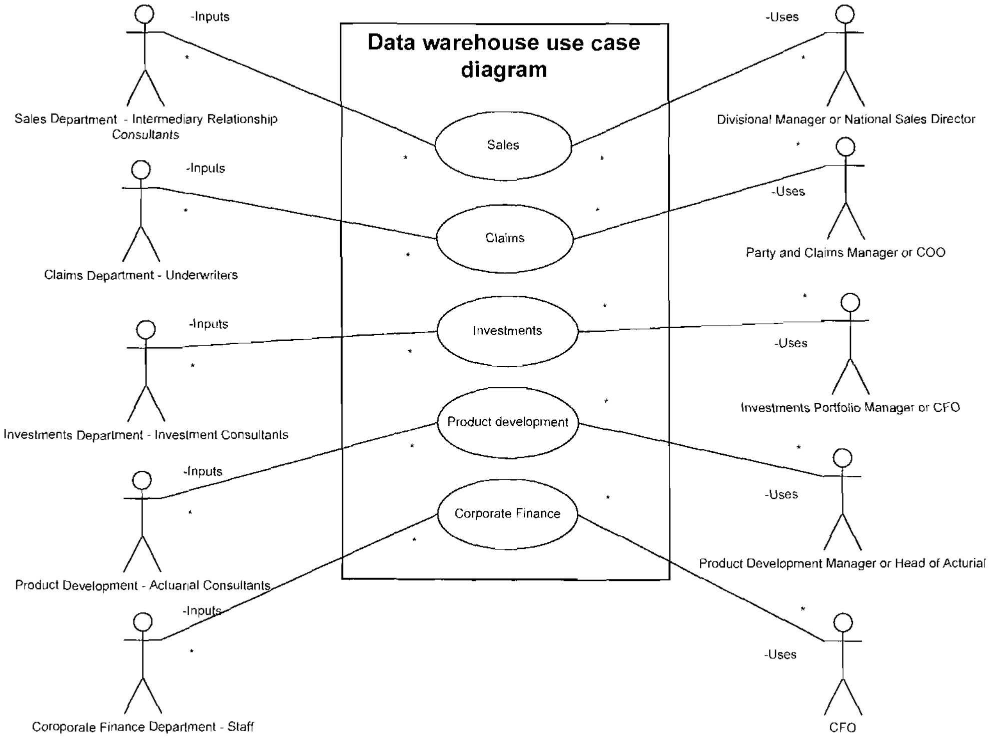

-Inputs
Sales Department - Intermediary Relationship Consultants
-Inputs
Claims Department - Underwriters
-Inputs
Investments Department - Investment Consultants
-Inputs
Product Development - Actuarial Consultants
-Inputs
Coroporate Finance Department - Staff
Data warehouse use case diagram
Sales
Claims
Investments
Product development
Corporate Finance
-Uses
Divisional Manager or National Sales Director
-Uses
Party and Claims Manager or COO
-Uses
Investments Portfolio Manager or CFO
-Uses
Product Development Manager or Head of Acturial
-Uses
CFO

Figure 7-4 Data warehouse use case diagram for the insurance company

Figure 7-4 illustrates the following facts:

- The intermediary relationship consultants are responsible for providing the input to sales. The divisional manager and national sales director use the sales figures.
- The underwriters are responsible for providing the input to claims. The party and claims manager and chief of operations director (COO) use the claims figures.
- The investment managers are responsible for providing the input to investments. The investment portfolio managers and the chief financial officer (CFO) use the investment figures.

- The actuarial consultants are responsible for providing the input to product development. The product development manager and head of actuaries use the product development figures.
- The finance clerks are responsible for providing the input to corporate finance. The chief financial officer (CFO) uses the corporate finance figures.

The above facts are summed in Table 7-1.

DW Use Case Number	**Department**	**Party responsible for providing information**	**Users**
DWUC01	Sales department	intermediary relationship consultants	Divisional manager and national sales director
DWUC02	Claims	Underwriters	Party and claims manager and chief of operations director (COO)
DWUC03	Investments	Investment managers	Investment portfolio managers and the chief financial officer (CFO)
DWUC04	Claims	Actuarial consultants	Claims manager, Finance manager
DWUC05	Finance	Product development	Product development manager and head of actuaries
DWUC06	Finance clerks	Corporate finance	Chief financial officer (CFO)

Table 7-1 List of data warehouse use cases.

A second session was held with the application support department to determine the scope and phases of the data warehouse. The outcome of this meeting determined the following:

- The data warehouse should support the following departments and should be considered as phase 1:
  - Sales
- The following departments should be considered as future phases of the data warehouse development:
  - Claims
  - Investments
  - Product development

- Corporate finance

Data warehouse essential use case

Resulting from the discussion, the scope of the project was based on data warehouse use case 1 (DWUC01). The following is the DW use case created from the DW use case diagram. Table 7-2 illustrates the summary of DWUC01.

DW Use Case: Sales (DWUC01) version 1.3		
Brief Description	This use case contains the high-level description of the sales department's inputs and outputs.	
Business function	Sales department is responsible for the following.	
Inputs	**Item**	**Brief Description**
	Product	The product name
	Date	The date when product is sold
	Division	The division where the product is sold
	Division Manager	The manager of that division
	Area	The regional office name
	Area Manager	The manager of the regional office
	Consultant	The consultant representing the brokers
	Broker	The broker description
	Policyholder	The policyholder
Outputs	• Sales report	
Possible grains	• The sale of a product. (finest grain) • The sale per product (rolled up) • The sale of products per time line (rolled up) • The sale of products per region (rolled up) • The sale of products per consultant (rolled up)	

Table 7-2 DWUC01 - Sales

The following is a discussion on the business process use case diagram.

Business process use case diagram

The business process use case diagram required facilitated sessions with the manager of the sales department. Figure 7-5 illustrates the business processes within the sales department.

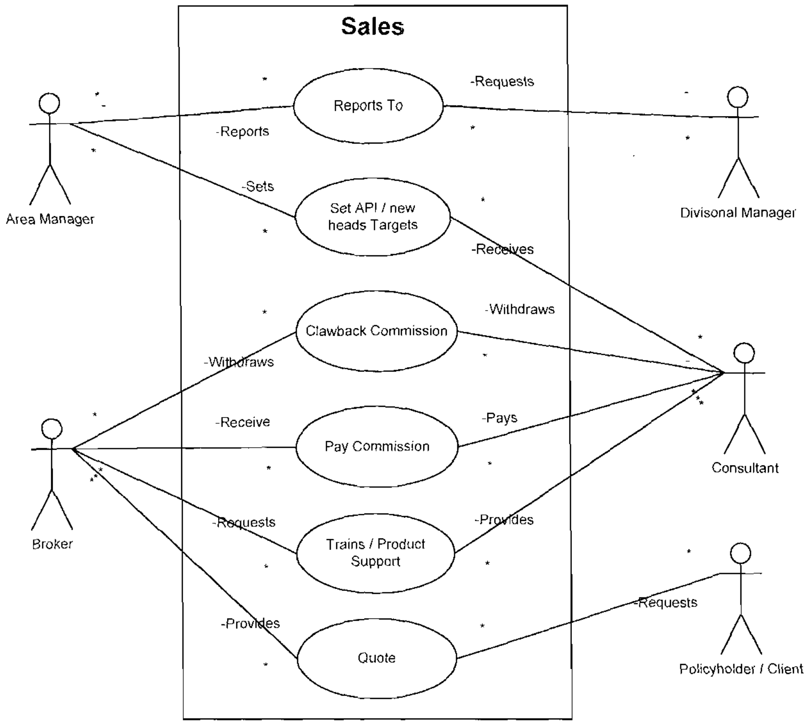


Figure 7-5 Business process diagram for Sales department

Based on the business process use case diagram in Figure 7-5, a list of use cases was created (illustrated in Table 7-3), which needed to be investigated further.

Use case number	Description
UC01	Area managers report to the divisional manager on sales key performance indicators (KPI).
UC02	Area manager sets annual performance indicators (API) and new head count per consultant.
UC03	Consultant claw back commission paid to broker based on policy agreement.
UC04	Consultant pays commission to broker based on policy agreement.
UC05	Consultant provides training and product support to broker.
UC06	Broker quotes the policyholder or client for insurance.

Table 7-3 list of use cases in sales busines process diagram

The following section is a discussion on the business process use cases already listed in Table 7-1.

Business process use cases

The following tables (Table 7-4 to Table 7-10) illustrate the business process (BP) use cases created from the list in Table 7-1.

A typical business process contains information about the business process identified, as well as information on the supporting mechanisms in the process, i.e. reports.

BP Use Case: Reports To (UC01) version 1.1		
Brief Description of business process	Area managers report to the divisional manager on sales key performance indicators (KPI).	
Actors	• Area manager • Divisional manager	
Precondition	Report requested	
Post condition	Report delivered	
Basic course of action	Divisional manager requests performance reports. The following reports are identified as performance reports. • Head count report • API report	
Report information	• Head count report - Date - Division - Divisional manager - Area - Area manager - Consultant - Head count - Target - Target YTD - Achieved - Achieved total - Total	• API report - Date - Division - Divisional manager - Area - Area manager - Consultant - Product - API count - Target - Target YTD - Achieved - Achieved total - Total

Table 7-4 Reports To business process use case

BP Use Case: Set API / heads (UC02) version 1.1	
Brief Description of business process	Area manager sets annual performance indicators (API) and new head count per consultant.
Actors	• Area manager • Consultant
Precondition	n/a
Post condition	New API and head target set.
Basic course of action	Area manager defines targets for consultants according to formula. Reports identified: • New target report

Table 7-5 Set API / Heads business process use case

Report information	• Head count report - Date - Division - Divisional manager - Area - Area manager - Consultant - Target achieved - New target	• API report - Date - Division - Divisional manager - Area - Area manager - Consultant - Product - Target achieved - New target
Formulas defined	• New head target formula - New target = target achieved * 1.2 - if new consultant then default is 100 • New API target formula - New target = target achieved * 1.3 - if new consultant then default is 2500	

Table 7-6 (Continued) Set API / Heads business process use case

BP Use Case: Claw back Commission (UC03) version 1.1	
Brief Description of business process	Consultant claw back commission paid to broker based on policy agreement.
Actors	• Consultant • Broker
Precondition	Policyholder cancels policy within 2 years of the issue date of policy.
Post condition	Claw back commission
Basic course of action	Consultant claw back commission from broker. Reports identified: • Claw back report
Report information	• Claw back report - Date - Division - Divisional manager - Area - Area manager - Consultant - Broker - Policy number - Product - Amount

Table 7-7 Claw back commission business process use case

BP Use Case: Pay Commission (UC04) version 1.1	
Brief Description of business process	Consultant pays commission to broker based on policy agreement.
Actors	• Consultant • Broker
Precondition	Policyholder needs to take out a policy.
Post condition	Paid commission
Basic course of action	Consultant pays broker commission Reports identified: • Commissions report
Report information	• Commissions report Date - Division - Divisional manager - Area - Area manager - Consultant - Broker - Policy number - Product - Amount for Year 1 - Amount for Year 2 (if applicable)

Table 7-8 Pay commission business process use case

BP Use Case: Training (UC05) version 1.1	
Brief Description of business process	Consultant provides training and product support to broker.
Actors	• Consultant • Broker
Precondition	Broker needs to be registered with a financial services provider and should not have any mandates with the insurance company.
Post condition	Mandate to sell products
Basic course of action	Broker needs to register on learning site. Once registered the broker needs to work through the guides and assignments The broker needs to pass the required tests on each product to get a mandate to sell products. Report identified: • Broker test report
Report information	• Broker test report - Date - Broker - Test number - Product name - Score

Table 7-9 Training business process use case

BP Use Case: Quote (UC06) version 1.1	
Brief Description of business process	Broker quotes the policyholder or client for insurance.
Actors	• Broker • Policyholder
Precondition	Broker needs to be registered with a financial services provider and should have the required mandate with the insurance company.
Post condition	Quote for insurance
Basic course of action	Policyholder request quote for insurance to broker. The broker uses an online application to quote for the required insurance The policyholder accepts or rejects the quote. Reports identified • Product quote report
Report information	• Product quote report Broker - Date - Broker - Product name - Sum Assured - Premium

Table 7-10 Quote business process use case

User interface prototyping

Section 6.3.1.3 determines that user interfaces should be prototyped and that they can be in the form of:

- Reports (on the department)
- End user desktop applications
- Data mining models
- Downstream operational systems

No further user interfaces other than reports were identified in the requirements sessions. The reports identified are listed in Table (Continued) 7-12.

Report name	**Report fields**
Sales Stats report	- Date sold - Division - Divisional manager - Sales area - Sales area manager - Consultant - Broker - Product

Table 7-11 List of reports identified

Report name	Report fields
	- API - New Head - Target API - Target Head - Target API YTD - Target YTD Head - Achieved API - Achieved Head
Claw back report	- Claw back date - Division - Divisional manager - Area - Area manager - Consultant - Broker - Policy number - Product - Amount
Commission Incentive report	- Date - Division - Divisional manager - Area - Area manager - Consultant - Broker - Policy number - Product - Amount for Year 1 - Amount for Year 2 (if applicable)

Table (Continued) 7-12 List of reports identified

The lists of reports identified above (Table (Continued) 7-12), were documented in the business process use cases. The next activity was to create the domain model. The following is a discussion of the domain model.

Domain modelling

As described in section 6.3.1.4, domain modelling seeks to identify classes and objects common to all applications within the domain (Booch, 1994:157). The concepts are derived from nouns and noun pareses in the business essential use cases and the data warehouse user interface prototypes.

Class Responsibility Collaborator Cards (CRC) is a useful technique for discovering classes representing concepts. This technique should be used in a brainstorming session.

It is also explained in section 6.3.1.4 that the CRC for DW development should be referred to as DW CRC, thereby avoiding confusion between operational systems (CRC) and DW systems (DW CRC). The business CRCs created, are listed from Table 7-13to Table 7-22.

CRC: Policyholder (or Insured) (Version 1.2)	
Class of policyholder Initials Last name First names Date of birth First Language Disposable Income Education level Employment status Gender Marital Status Ethnicity Is doer of occupations Is doer of habit Is subject of medical conditions Is subject of assessment results External reference Contact Preferences	(no actions)

Table 7-13 Policyholder CRC

CRC: Broker (Version 1.2)	
Initials Last name First names Date of birth First Language Gender Marital Status Ethnicity External reference Contact Preferences Reports to consultant	(no actions)

Table 7-14 Broker CRC

CRC: Consultant (Version 1.1)	
Initials Last name First names Date of birth First Language Gender Ethnicity External reference Contact Preferences Belongs to area Reports to area manager API Target Head Count Target	(no actions)

Table 7-15 Consultant CRC

CRC: Area Manager (Version 1.1)	
Initials Last name First names Date of birth First Language Gender Ethnicity External reference Contact Preferences Belongs to division Reports to divisional manager	(no actions)

Table 7-16 Area Manager CRC

CRC: Divisional Manager (Version 1.1)	
Initials Last name First names Date of birth First Language Gender Ethnicity External reference Contact Preferences Belongs to insurance company	(no actions)

Table 7-17 Divisional Manager

CRC: Product (Version 1.4)	
Start Date End Date External reference Product Kind Product Components Product Properties Product Roles (include the sum assured and premium)	(no actions)

Table 7-18 Product CRC

CRC: Quote and Statement of benefits (Version 1.1)	
Policyholder external reference Broker external reference Product external references	(no actions)

Table 7-19 Quote and statement of benefits CRC

CRC: Learn Student module (Version 1.1)	
Broker external reference Module description Assignment date Evaluation date Points earned Result	(no actions)

Table 7-20 Learn student module CRC

CRC: Commission Paid(Version 1.2)	
Broker external reference Policyholder external reference Product start date Product external reference Product kind Product Sum Assured Product Premium Commission Year 1 Commission Year 2	(no actions)

Table 7-21 Commission Paid

CRC: Commission claw back(Version 1.1)	
Broker external reference Policyholder external reference Product end date Product external reference Product kind Product Sum Assured Product Premium Commission Year 1 Commission Year 2	(no actions)

Table 7-22 Commission claw back

The actor CRCs created is illustrated from Table 7-23 to Table 7-26.

Policyholder (or Insured) (Version 1.1)	<<Actor>>
Request for Insurance Provides party information Apply for insurance Cancel insurance	Broker Policyholder

Table 7-23 Policyholder actor CRC

Broker (Version 1.1)	<<Actor>>
Provides party (broker and policyholder) information Request quote for insurance Request training Provide commission incentive received Do commission claw back Reports to	Broker Policyholder Quote Application Training Application Commission Paid Commission claw back Consultant

Table 7-24 Broker actor CRC

Consultant (Version 1.0)	<<Actor>>
Provides party (consultant) information Pays commission incentive Claw back commission	Consultant Commission Paid Commission claw back

Table 7-25 Consultant actor CRC

Area Manager (Version 1.1)	<<Actor>>
Provides party (area manager) information Set targets Reports sales stats	Area manager Consultant Sales Stats

Table 7-26 Area manager actor CRC

The user interface CRCs (all in the form of reports), are illustrated in Table 7-27 to Table 7-29.

Sales Stats (Version 1.3)	**<<Report>>**
Date sold Division Divisional manager Sales area Sales area manager Consultant Broker Product API New Head Target API Target Head Target API YTD Target YTD Head Achieved API Achieved Head	Policyholder Broker Consultant Area manager Divisional manager Quote and Statement of benefits

Table 7-27 Sales stats report CRC

Commission Incentive (Version 1.3)	**<<Report>>**
Date Division Divisional manager Sales area Sales area manager Consultant Broker Product Policyholder Product start date Product external reference Product kind Product Sum Assured Product Premium Commission Year 1 Commission Year 2	Policyholder Broker Consultant Area manager Divisional manager Quote and Statement of benefits Commission Incentive paid

Table 7-28 Commission incentive report CRC

Commission claw back (Version 1.4)	**<<Report>>**
Date	Policyholder
Division	Broker
Divisional manager	Consultant
Sales area	Area manager
Sales area manager	Divisional manager
Consultant	Quote and Statement of benefits
Broker	Commission claw back
Product	
Policyholder	
Product start date	
Product end date	
Product external reference	
Product kind	
Product Sum Assured	
Product Premium	
Commission Year 1	
Commission Year 2	

Table 7-29 Commission claw back report CRC

Supplementary documentation

Section 6.3.1.5 requires the business rules to be investigated. For this study, only the business rules applicable to reporting were investigated. Business rules eligible for members and products were not investigated, as these rules form part of the company's production line. Table 7-30 to Table 7-34 illustrate the rules applicable to the CRCs created (Table 7-13 - Table 7-29).

BRRV01 – External reference	
Description of Rule: Determines the nature of the external reference	
Test Parameters	1. External reference
Condition for which Rule is True	If external reference start with a numeric number it can be linked to the following: • Broker agreement • Consultant agreement • Area Manager agreement If external reference start with a date i.e. 20010101 then it qualifies as a product: • Sick pay and permanent incapacity product – follows a 01 after the date and 4 digits. i.e. 20010101011234 • Life assurance follows a 02 after the date and four digits i.e. 20010101021234 • Health assurance follows a 03 after the date and four digits i.e. 20010101031234 • Retirement annuity follows a X after the date and four digits i.e. 20010101X1234
Category	Report Validity

Table 7-30 External reference business rule

BRSH01 – Broker validity	
Description of Rule: Broker can only report to one consultant	
Test Parameters	1. External reference (of broker agreement)
Condition for which Rule is True	If external reference is a broker
Category	Sales hierarchy

Table 7-31 Broker validity business rule

BRSH02 – Consultant validity	
Description of Rule: Consultant can only report to one area manager and have multiple brokers.	
Test Parameters	1. External reference (of consultant agreement)
Condition for which Rule is True	If external reference is a consultant
Category	Sales hierarchy

Table 7-32 Consultant validity business rule

BRSH03 – Area manager validity	
Description of Rule: Area manager can only report to one divisional manager and have multiple consultants.	
Test Parameters	1. External reference (of area manager)
Condition for which Rule is True	If external reference is a area manager
Category	Sales hierarchy

Table 7-33 Area manager validity rule

BRSH04 – Area	
Description of Rule: Area can have only one area manager	
Test Parameters	1. External reference (of area manager agreement)
Condition for which Rule is True	If area manager then only one assigned area
Category	Sales hierarchy

Table 7-34 Area business rule

BRRC01 – Target YTD Calculation	
Description of Rule: Target	
Test Parameters	1. Month 2. Target
Calculation definition rule	Target YTD = Target / 12 * Month i.e. 1000 / 12 * 3 (for March) = 250
Category	Report Calculation

Table 7-35 Target Year To Date Calculation business rule

BRRC02 – Achieved Calculation	
Description of Rule: Target Achieved	
Test Parameters	1. Month 2. Target 3. Actual figure (API or Head)
Calculation definition rule	Actual figure / (Target / 12 * Month) * 100 i.e. 200 / (1000 / 12 * 3 (for March)) *100 = 80%
Category	Report Calculation

Table 7-36 Achieved Calculation business rule

Outline on the DW maintenance and growth

DW maintenance and growth determines the protocol followed for maintaining and developing the DW. In the following sections, DW maintenance and growth is discussed.

Debugging

Any bug detected on the system should be logged as a bug at the service desk. A service desk number will be issued. A bug is defined as any existing development not functioning correctly.

Change Request

Any change request for the system should be logged as a change request at the service desk. A change request number will be issued. A change request is defined as any new development needed to change existing functionality or introduce new functionality.

All bugs and change requests need to be tested on the following environments:

- Staging – once the developers involved are happy with the development, it should be promoted to quality assurance (QA).
- QA – once business testing is completed, it can be promoted to production.
- Production – all production changes will require approval from both the business owner and application support manager.

Source control

All development will be stored in a repository, while two streams of source control are used:

- Development stream – contains all the source code and designs for development.
- Head stream – contains all the source code and designs for development approved by business and application support.

In the following section, dimensional modelling for the DW is discussed.

7.6.3. Dimensional modelling

Section 6.3.2 explains the dimensional modelling phase. An analysis and design activity allows for a typical object-oriented development approach. The following discussion will concentrate firstly on the analysis and secondly on the design of the dimensional modelling.

Phase	Activity
Object Oriented Analysis	• System Use Case • Sequence Diagram • Conceptual Class Modelling • Activity Diagram • User Interface Prototyping • Supplementary Specifications • User Documentation • Organise Packages

Figure 7-6 Object-Oriented Analysis diagram

7.6.3.1. Dimensional modelling analysis – DW System use case

Up to this point two types of use cases have been developed, i.e.:

- Data warehouse essential use cases
- Business process essential use cases

According to Section 6.3.2.1, the DW system use case contains more system dependent information. This study focused on one DW use case, namely DWUC01 illustrated in Table 6-1. Table 7-38 shows the DW systems use case created for DWUC01.

DW System Use Case: Sales (System - DWUC01) version 1.3	
Brief Description	This use case contains the high-level description of the sales department's inputs and outputs.
Business function	Sales department is responsible for the following

Table 7-37 System DWUC01 - Sales

Inputs	**Item**	**DataType**	**Brief Description**
	Product	Product Type	The product name
	Date	Date	The date when product is sold
	Division	Varchar(255)	The division where the product is sold
	Divisional Manager	PartyType	The manager of that division
	Area	Varchar(255)	The regional office name
	Area Manager	PartyType	The manager of the regional office
	Consultant	PartyType	The consultant representing the brokers
	Broker	PartyType	The broker description
	Policyholder	PartyType	The policyholder
	Money Provision	Currency	Contains the premium and sum assured
Input Source	Production databases: • SPF • PARTY • FTX		
Outputs	Sales Stats (Version 1.3) Commission Incentive (Version 1.3) Commission claw back (Version 1.4) Formats for the reports should be in CSV and PDF		
Possible grains	Product level		

Table 7-38 (Continued) System DWUC01 - Sales

The data types illustrated in Table 7-38 contain both primitive and custom data types. The customer data types are defined as shown in Table 7-39 to Table 7-42.

Data Type Definition – ProductType			
Brief Description	Contains the product data agreement		
ProductType	**Item**	**DataType**	**Brief Description**
	StartDate	Date	Start date of the product
	EndDate	Date	The date when product is sold
	Kind	Varchar(255)	Is the kind of product
	LifeCycleStatus	Int	Defines that status of the product.
	Properties	PropertyType[]	Array of PropertyType
	ExternalReference	Varchar(255)	External Reference
	Roles	PartyType[]	Array of partyType
	Components	ProductType[]	Array of ProductType[]

Table 7-39 Data type definition of ProductType

Data Type Definition – PropertyType			
Brief Description	Contains the property type data		
ProductType	**Item**	**DataType**	**Brief Description**
	Kind	Varchar(255)	Is the kind of the property
	Value	Varchar(255)	Property value

Table 7-40 Data type definition of PropertyType

Data Type Definition – PartyType			
Brief Description	Contains the party agreement data		
ProductType	**Item**	**DataType**	**Brief Description**
	Class	Varchar(255)	Specifies that class of party
	BirthDate	Date	Date of birth
	Language	Varchar(255)	Primary language of party
	DisposableIncome	Double	Amount of income.
	EducationLevel	Int	Level of education (0-6)
	EmploymentStatus	Varchar(255)	Employment status
	MaritalStatus	Varchar(255)	Marital status
	Ethnicity	Varchar(255)	Ethnicity of party
	Occupations	Varchar[](255)	Occupations of party
	DefaultName	Varchar(255)	Default name of party
	FirstNames	Varchar[](255)	Names of party
	LastName	Varchar(255)	Last name of party
	DefaultContactPreference	ContactPreference	The default contact preference of party.
	ContactPreferences	ContactPreference[]	Array of ContactPreferences

Table 7-41 Data type definition of PartyType

Data Type Definition – ContactPreference			
Brief Description	Contains the contact preference data type		
ProductType	**Item**	**DataType**	**Brief Description**
	Kind	Varchar(255)	Kind of contact preference
	Address	Varchar(255)	If contact preference is an address.
	PhoneNumber	Varchar(255)	If contact preference is a phone number.
	EmailAddress	Varchar(255)	If contact preference is an email address.

Table 7-42 Data type definition of ContactPrefrence

7.6.3.2. Dimensional modelling analysis – Business process system use case

Section 6.2.2.2 explains that the business process essential use case model is evolved into a system use case. It is similar to the business process essential use case with the exception that it includes high-level implementation decisions, such as the screen numbers and properties, as well as includes and inheritance.

The business process systems use cases are defined in Table 7-43 to Table 7-49.

BP Use Case: Reports To (Systems - UC01) version 1.0	
Brief Description of business process	Area managers report to the divisional manager on sales key performance indicators (KPI).
Actors	• Area Manager • Divisional manager
Precondition	Report requested
Post condition	Report delivered
Basic course of action	Divisional manager requests performance reports. The following reports are identified as performance reports. • Head count report • API report
Report information	Report defined in Report CRC as Sales Stats (Version 1.3)

Table 7-43 Reports To business process systems use case

BP Use Case: Set API / heads (Systems - UC02) version 1.0	
Brief Description of business process	Area manager sets annual performance indicators (API) and new head count per consultant.
Actors	• Area Manager • Consultant
Precondition	n/a
Post condition	New API and head target set.
Basic course of action	Area manager defines targets for consultants according to formula. Reports identified: • New target report
Report information	Report defined in Report CRC as Sales Stats (Version 1.3)
Formulas defined	New head/API target formula defined in BRRC01

Table 7-44 Set API / Heads business process systems use case

BP Use Case: Claw back Commission (Systems - UC03) version 1.0	
Brief Description of business process	Consultant claw backs commission paid to broker based on policy agreement.
Actors	• Consultant • Broker
Precondition	Policyholder cancels policy within 2 years of the issue date of policy.
Post condition	Claw back commission

Table 7-45 Claw back commission business process systems use case

Basic course of action	Consultant claw back commission from broker. Reports identified: • Claw back report
Report information	Report defined as Commission claw back (Version 1.4)

Table 7-46 (Continued) Claw back commission business process systems use case

BP Use Case: Pay Commission (Systems - UC04) version 1.0	
Brief Description of business process	Consultant pays commission to broker based on policy agreement.
Actors	• Consultant • Broker
Precondition	Policyholder needs to take out a policy.
Post condition	Paid commission
Basic course of action	Consultant pays broker commission Reports identified: • Commissions report
Report information	Report defined as Commission Incentive (Version 1.3)

Table 7-47 Pay commission business process systems use case

BP Use Case: Training (Systems - UC05) version 1.0	
Brief Description of business process	Consultant provides training and product support to broker.
Actors	• Consultant • Broker
Precondition	Broker needs to be registered with a financial services provider and should not have any mandates with the insurance company.
Post condition	Mandate to sell products
Basic course of action	Broker needs to register on learning site. Once registered, the broker needs to work through the guides and assignments The broker needs to pass the required tests on each product to get a mandate to sell products. Report identified: • Broker test report
Report information	• Broker test report - Date - Broker - Test number - Product name - Score

Table 7-48 Training business process use case

BP Use Case: Quote (Systems - UC06) version 1.0	
Brief Description of business process	Broker quotes the policyholder or client for insurance.
Actors	• Broker • Policyholder
Precondition	Broker needs to be registered with a financial services provider and should have the required mandate with the insurance company.
Post condition	Quote for insurance
Basic course of action	Policyholder requests quote for insurance to broker. The broker uses an online application to quote for the required insurance The policyholder accepts or rejects the quote. Reports identified • Product quote report
Report information	• Product quote report Broker - Date - Broker - Product name - Sum Assured - Premium

Table 7-49 Quote business process use case

7.6.3.3. Dimensional modelling analysis – Sequence diagrams

Section 6.3.2.3 explains that the function of the data warehouse use cases is to provide an overview of the company (the “big picture”) and not the process flow of the business. Owing to this, it will be senseless to create a sequence diagram from the data warehouse use cases, as the function of a sequence diagram is to model the interaction between classes in a process.

For the purpose of dimensional modelling, the logic flow is not all that important, however, it is important to discover which classes interact with one another and what that interaction entails.

The sequence diagrams are illustrated in Figure 7-7 to Figure 7-10.

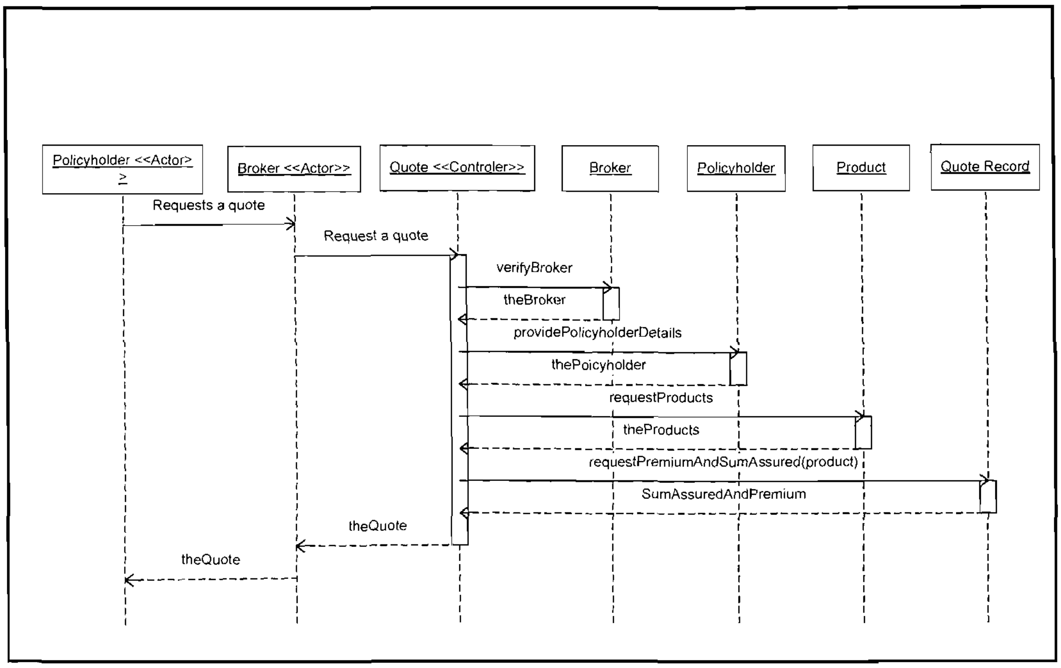


Figure 7-7 Quote Sequence Diagram

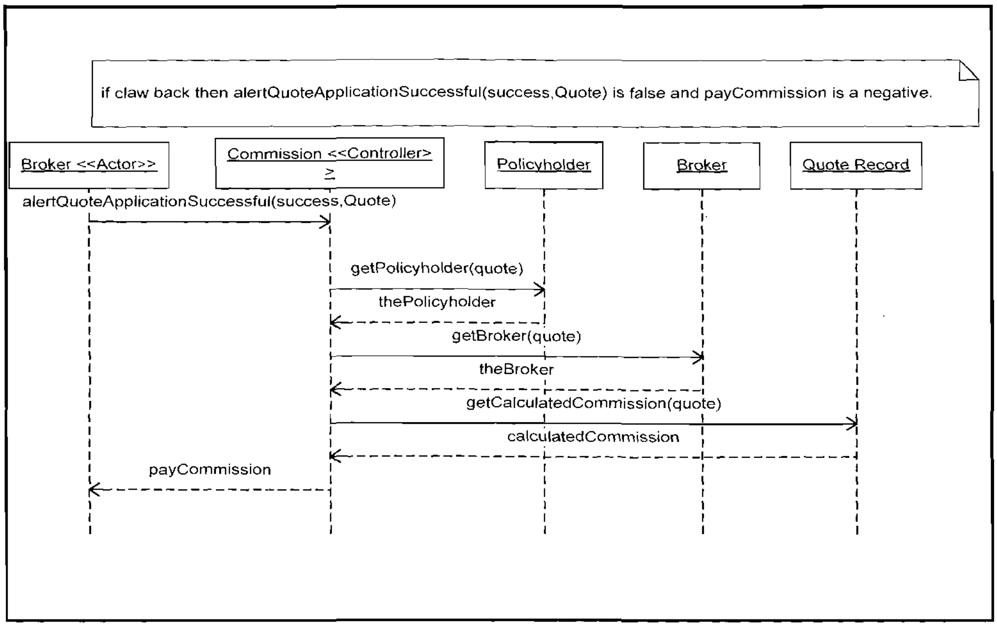

if claw back then alertQuoteApplicationSuccessful(success,Quote) is false and payCommission is a negative.
Broker <<Actor>>
Commission <<Controller>
>
Policyholder
Broker
Quote Record
alertQuoteApplicationSuccessful(success,Quote)
getPolicyholder(quote)
thePolicyholder
getBroker(quote)
theBroker
getCalculatedCommission(quote)
calculatedCommission
payCommission

Figure 7-8 Commission Sequence Diagram

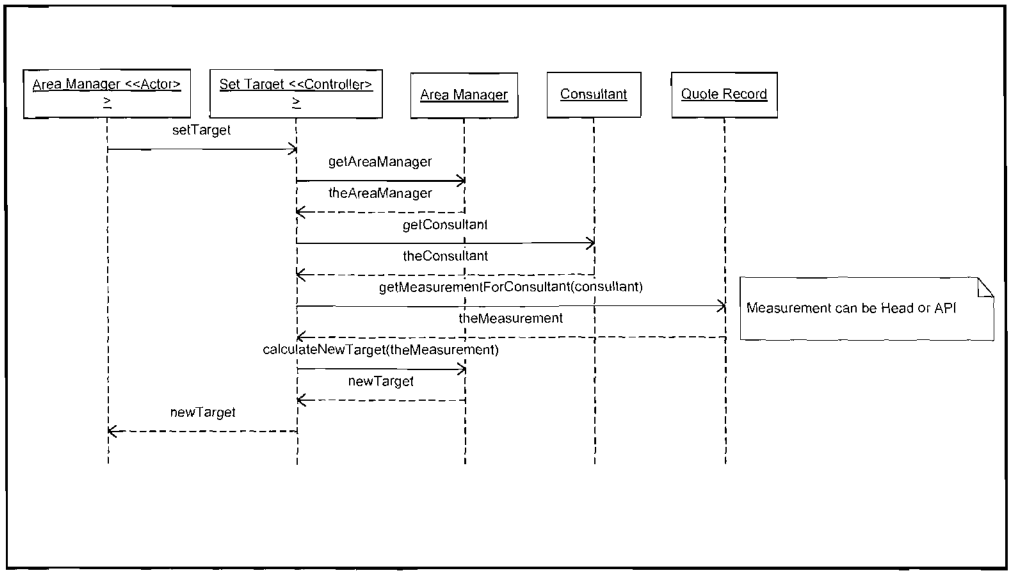


Figure 7-9 Set Target Sequence Diagram

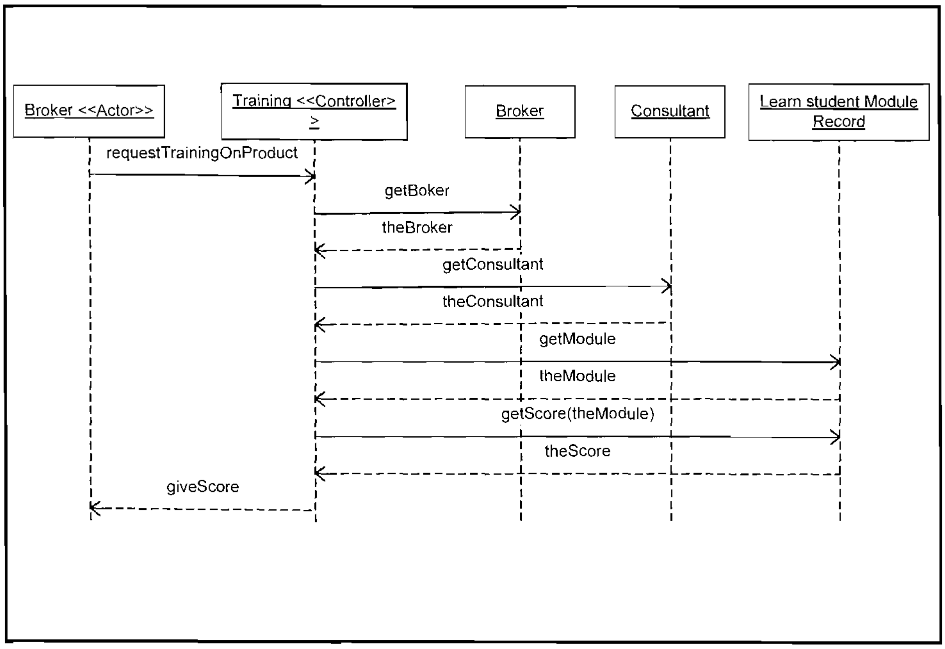


Figure 7-10 Training Sequence Diagram

7.6.3.4. Dimensional modelling analysis – Data Warehouse Bus Architecture Matrix

Section 6.3.2.4 explains that the business use cases and the DW use cases should be combined to get a better understanding of the business. Table 7-50 illustrates the combination of DWUC01 (DW use case) with the business use cases.

DW Use Case Number	Department	Business Use Case Number	Business Process
DWUC01	Sales department	UC01	Reports To
		UC02	Set API / heads
		UC03	Claw back Commission
		UC04	Pay Commission
		UC05	Training
		UC06	Quote

Table 7-50 Combination of the DW use cases with the business use cases

From Table 7-50, one can derive the following conclusions:

- UC01 – Reports To is not a business process that no data can be captured on, thus it will not be modeled in the data warehouse.
- UC02 – Set API (Annual Performance Indicator)/ heads is a key performance indicator (KPI) for consultants, thus data needs to capture on this to provide the new API / head targets for the consultants.
- UC03 – Claw back Commission is done when the policyholder cancels the policy agreement prematurely. This can be seen as a negative entry on the broker's commission statement.
- UC04 – Pay Commission: When a policy is sold, a broker is paid a commission based on the amount of the sum assured of the policy agreement. This can be seen as a positive entry on the broker's commission statement.
- UC05 – Training should contain the score of the module of the broker.
- UC06 – Quote should contain the product, sum assured and the premiums for the requested quote.

Based on the above conclusions, the following subject areas can be created:

- Target Annual Performance Incentive (API) / Head
- Commission
- Training
- Quote

Claw back commission and pay commission are modeled together, because they are in essence the same type of transaction. The one is a negative inventive transaction and the other a positive incentive transaction. Training and quote are modeled as own entities.

The business classes that were identified are:

- Policyholder
- Broker
- Consultant
- Area Manager
- Divisional Manager
- Product
- Quote and Statement of benefits
- Learn student module
- Commission Paid
- Commission claw back

The actor classes that were identified are:

- Policyholder
- Intermediary
- Consultant
- Assistant Manager

The interface models that were identified are:

- Sales Stats <<Report>>
- Commission Incentive <<Report>>
- Commission claw back <<Report>>

Based on the above classes and analysis, the following dimensions were created:

- Policyholder
- Intermediary
- Consultant

- Actual
- Learn student module (only descriptive information)

Based on the links between the classes in the sequence diagrams created (shown in Figure 7-7 to Figure 7-10) and the above analysis, Data Warehouse Bus Architect Matrix can be created. Figure 7-11 represents the Data Warehouse Bus Architect Matrix for the case study.

	Time	**Policyholder**	**Intermediary**	**Consultant**	**Actual**	**Learn Student Module**
Target API / Head	√			√		
Target API	√	√	√	√	√	
Target Head	√		√	√		
Commission	√	√	√	√	√	
Training	√	√		√		√
Quote	√	√	√	√	√	

Figure 7-11 Data Warehouse Bus Architect Matrix for the case study

While having a bird's eye view of what the data warehouse should contain, business at this stage decided to narrow the scope for the development of the data warehouse to only the highlighted subject areas and dimensions. The development of the data marts followed the lifecycle illustrated in Figure 7-12.

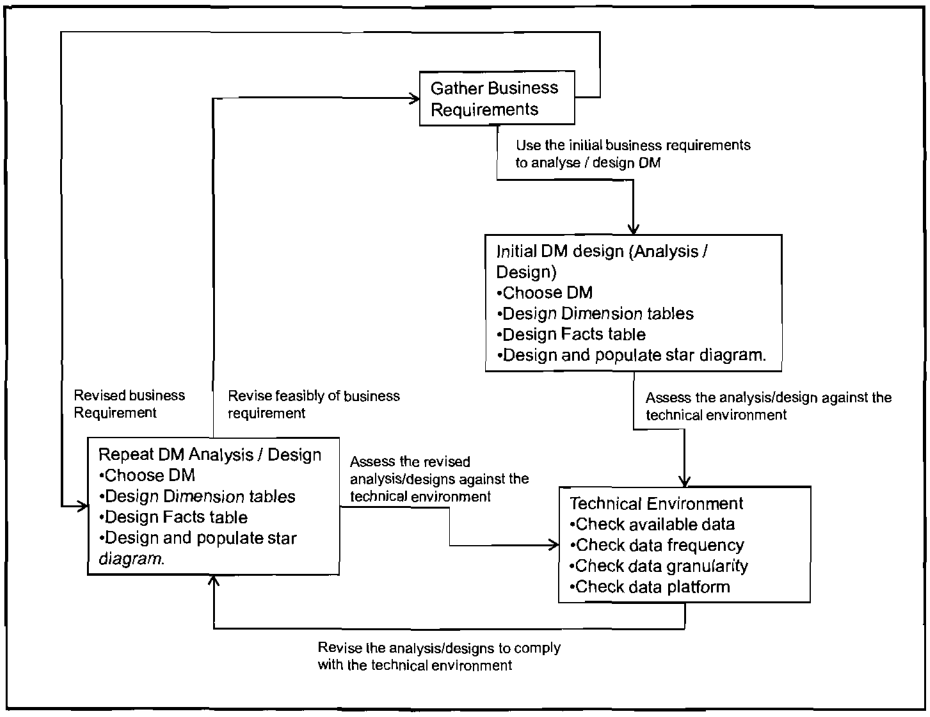

Gather Business Requirements
Use the initial business requirements to analyse / design DM
Initial DM design (Analysis / Design)
•Choose DM
•Design Dimension tables
•Design Facts table
•Design and populate star diagram.
Revised business Requirement
Revise feasibly of business requirement
Assess the analysis/design against the technical environment
Repeat DM Analysis / Design
•Choose DM
•Design Dimension tables
•Design Facts table
•Design and populate star diagram.
Assess the revised analysis/designs against the technical environment
Technical Environment
•Check available data
•Check data frequency
•Check data granularity
•Check data platform
Revise the analysis/designs to comply with the technical environment

Figure 7-12 Lifecycle of a DM development

The first iteration of the analysis and design was based on the initial requirements. From the analysis / design, an assessment of the technical environment was done to evaluate the feasibility of the current analysis / design. Based on the evaluation, the analysis / design was changed accordingly (an iterative process). Some of the requirements, being the membership type on IAA (source system), had to be reconsidered.

The next section covers the analysis and design for the Target API subject area.

7.6.3.5. Dimensional modelling analysis – Dimension table detail

Section 6.3.2.6 explains that the dimension table diagram needs to be completed for each dimension. It illustrates the grain of each dimension, as well as the cardinality of each dimension attribute, with a top down view of all the hierarchies (Kimball *et al.*, 1998:281).

The following dimension tables were designed:

- Date
- Policyholder

- Intermediary
- Consultant
- Actual

Table 7-52 to Table 7-57 indicate the dimension hierarchy and attribute details.

Dimension Table: Date version 1.0		
Hierarchy	Top Level	YEAR
	Level 1	SEMESTER
	Level 2	QUARTER
	Level 3	MONTH
	Level 4	DAY

Table 7-51 Time dimension hierarchy and attribute detail

Attribute Detail				
Attribute Name	**Attribute Description**	**Cardinality**	**Slowly Changing Dimension Policy**	**Sample Value**
YEAR	Indicates the calendar year	10	Not updated	2007
SEMESTER	Indicates the calendar semester	2	Not updated	1
QUARTER	Indicates the calendar quarter	4	Not updated	1
MONTH	Indicates the calendar month	12	Not updated	1
DAY	Indicates the calendar day	365	Not updated	1
DISPLAY_DATE	Full date in yyyy-mm-dd	365	Not updated	2007-01-01
DATE	Full date in datetime format	365	Not updated	2007-01-01
DAY_NAME	Name of the day	7	Not updated	Monday
DAY_OF_WEEK	1-Monday to 7-Sunday	7	Not updated	1
WEEK_OF_YEAR	Week of the year 1 to 52	52	Not updated	1

Table 7-52 (Continued) Time dimension hierarchy and attribute detail

Dimension Table: Policyholder version 1.0				
Hierarchy	Top Level	N/A		
Attribute Detail				
Attribute Name	**Attribute Description**	**Cardinality**	**Slowly Changing Dimension Policy**	**Sample Value**
MEMBER_NUMBER	Indicates the member number	*	Type 1	1249954
FIRST_NAME	Indicates the first name	*	Type 1	Ralph
MIDDLE_NAMES	Indicates the middle names	*	Type 1	Reeves
LAST_NAME	Indicates the last name	*	Type 1	Kimball
DATE_OF_BIRTH	Indicates the birth date	365	Type 1	1950-01-01
LANGUAGE	Indicates the language	2	Type 1	English
GENDER	Indicates the gender	2	Not updated	Male
ETHNICITY	Indicates the ethnicity	5	Not updated	1

Table 7-53 Policyholder dimension hierarchy and attribute detail

Dimension Table: Intermediary version 1.0				
Hierarchy	Top Level	N/A		
Attribute Detail				
Attribute Name	**Attribute Description**	**Cardinality**	**Slowly Changing Dimension Policy**	**Sample Value**
MEMBER_NUMBER	Indicates the member number	*	Type 1	1249954
FIRST_NAME	Indicates the first name	*	Type 1	David
MIDDLE_NAMES	Indicates the middle names	*	Type 1	Reeves
LAST_NAME	Indicates the last name	*	Type 1	Avison
DATE_OF_BIRTH	Indicates the birth date	365	Type 1	1950-01-01

Table 7-54 Broker dimension hierarchy and attribute detail

LANGUAGE	Indicates the language	2	Type 1	English
GENDER	Indicates the gender	2	Not updated	Male
FSP	Indicates the financial service provider company	*	Type 1	Insurance Brokers A PTY(Ltd)
REPORTS_TO	Indicates the consultant that the broker reports to	*	Type 1	Consultant, B
BELONGS_TO	Indicates the region that the broker works under	*	Type 1	GAUTENG NORTH
AGREEMENT_TYPE	Type of broker	2	Type 1	Insurance Broker

Table 7-55 (Continued) Broker dimension hierarchy and attribute detail

Dimension Table: Consultant version 1.0				
Hierarchy	Top Level	N/A		
Attribute Detail				
Attribute Name	**Attribute Description**	**Cardinality**	**Slowly Changing Dimension Policy**	**Sample Value**
MEMBER_NUMBER	Indicates the member number	*	Type 1	1249954
FIRST_NAME	Indicates the first name	*	Type 1	David
MIDDLE_NAMES	Indicates the middle names	*	Type 1	Reeves
LAST_NAME	Indicates the last name	*	Type 1	Avison
DATE_OF_BIRTH	Indicates the birth date	365	Type 1	1950-01-01
LANGUAGE	Indicates the language	2	Type 1	English
GENDER	Indicates the gender	2	Not updated	Male
REPORTS_TO	Indicates the area manager that the consultant reports to	*	Type 1	Manager, A
BELONGS_TO	Indicates the region that the consultant works under	*	Type 1	GAUTENG NORTH
TARGET API	Indicates the required API	*	Type 1	10000
TARGET_HEAD	Indicates the required Head	*	Type 1	200

Table 7-56 Consultant dimension hierarchy and attribute detail

Dimension Table: Actual version 1.0

Hierarchy	Top Level	N/A		
Attribute Detail				
Attribute Name	**Attribute Description**	**Cardinality**	**Slowly Changing Dimension Policy**	**Sample Value**
PRODUCT_NAME	Indicates the name of the product	*	Type 1	Life Insurance
PRODUCT_BENEFIT	Indicates the benefit taken in the product.	*	Type 1	Live Cover

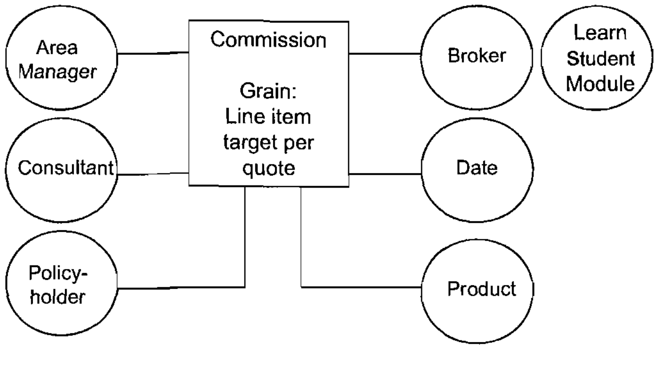
Table 7-57 Product dimension hierarchy and attribute detail

7.6.3.6. Dimensional modelling analysis – Fact table diagram

Section 6.3.2.5 explains that the fact table diagram illustrates the specific fact table and its context and also serves as an overview of all the dimensions that have been identified. Figure 7-13 illustrates both the fact table diagram and fact table detail diagram for the case matrix defined in Figure 7-11.

Fact Table: Target API (version 1.0)

Fact table diagram



Fact table detail diagram

FACT_API	
PK	ID
FK5 FK4 FK2 FK3 FK1 FK6	PARTY_INTERMEDIARY_ID PARTY_CONSULTANT_ID DIM_DATE PARTY_AREA_MANAGER_ID ACTUAL_ID PARTY_POLICYHOLDER_ID SOURCE_SYSTEM_ID SOURCE_SYSTEM_EXTRACT_DATE EXTRACT_DATE TRANSACTION_AMOUNT ALLOCATED_AMOUNT MEMBER_AGE COMMISSION_TYPE MONTHS_IN_FORCE ORIGINAL_REQUEST_DATE REQUESTED_DATETIME TRANSACTION_DATE

Figure 7-13 Fact table diagram and detail diagram for FACT_API

7.6.3.7. Dimensional modelling analysis – Identify sources

The data sources identified for the case study were only formal data sources and as such maintained by the IS department. Table 7-58 illustrates the data source information.

Data source definition version 1.1			
Source	**Business owner**	**Platform**	**Description**
Sales Logix	Application support	Windows 2003 / SQL Server 2000	Contains all sales data of intermediaries and their consultants.
IAA	Application support	Windows 2003 / SQL Server 2000	Maintains all party information and policy agreement administration.
Portfolio	Sales	Windows 2000 / XML	Quoting and member portfolio system.

Table 7-58 Data source definition for the data warehouse

7.6.3.8. Dimensional modelling analysis – Source to target mapping

Section 6.3.2.7 explains that the source to target mapping should be created once the data sources are defined. The source to target mapping for this case study is contained in Table 7-59 to Table 7-64.

Source to Target Mapping:Fact_API (Version 1.02)

Table Name	Column name	Data Type	Allow NULL	Target column description	Source System	Source DB / File	Data transform	Notes
FACT_API	ID	int	NO		new			gets created for each new entry
FACT_API	PARTY_INTERMEDIARY_ID	int	YES		DW			links to intermediary dimension
FACT_API	PARTY_CONSULTANT_ID	int	YES		DW			links to consultant dimension
FACT_API	DIM_DATE	varchar(8)	YES		DW			links to date dimension
FACT_API	PARTY_AREA_MANAGER_ID	int	YES		DW			links to area manager dimension
FACT_API	ACTUAL_ID	int	YES		DW			links to the policy agreement dimension
FACT_API	PARTY_POLICYHOLDER_ID	int	YES		DW			links to the policyholder dimension
FACT_API	SOURCE_SYSTEM_ID	int	YES		new			indicator to identify souce system extract
FACT_API	SOURCE_SYSTEM_EXTRACT_DATE	datetime	YES		new			date of extract
FACT_API	EXTRACT_DATE	datetime	YES		SLX		no	
FACT_API	TRANSACTION_AMOUNT	numeric(18,2)	YES		SLX		no	
FACT_API	ALLOCATED_AMOUNT	numeric(18,2)	YES		SLX		no	
FACT_API	MEMBER_AGE	int	YES		SLX		no	
FACT_API	COMMISSION_TYPE	varchar(255)	YES		SLX		no	
FACT_API	MONTHS_IN_FORCE	int	YES		SLX		no	
FACT_API	ORIGINAL_REQUEST_DATE	datetime	YES		SLX		no	
FACT_API	REQUESTED_DATETIME	datetime	YES		SLX		no	
FACT_API	TRANSACTION_DATE	datetime	YES		SLX		no	

Table 7-59 Source to Target for Fact API

Source to Target Mapping:DIM_PARTY_POLICYHOLDER (Version 1.06)

Table Name	Column name	Data Type	Allow NULL	Target column description	Source System	Source DB / File	Data transform	Notes
DIM_PARTY_POLICHOLDER	PARTY_POLICYHOLDER_ID	int IDENTITY(1,1)	NO		new			gets created for each new entry
DIM_PARTY_POLICHOLDER	SOURCE_SYSTEM_ID	int	YES		DW			Id of source system
DIM_PARTY_POLICHOLDER	SOURCE_SYSTEM_REFERENCE	varchar(255)	YES		DW			Reference on source system
DIM_PARTY_POLICHOLDER	SOURCE_SYSTEM_START_DT	datetime	YES		DW			Record start date of source system
DIM_PARTY_POLICHOLDER	SOURCE_SYSTEM_END_DT	datetime	YES		DW			Record end date of source system
DIM_PARTY_POLICHOLDER	SOURCE_SYSTEM_STATUS	varchar(255)	YES		DW			Indicator of source system status
DIM_PARTY_POLICHOLDER	IS_CURRENT	bit	YES		DW			Indicator for current record
DIM_PARTY_POLICHOLDER	PREVIOUS_ID	int	YES		DW			Points to previous record
DIM_PARTY_POLICHOLDER	RECORD_START_DT	datetime	YES		DW			Record start date
DIM_PARTY_POLICHOLDER	RECORD_END_DT	datetime	YES		DW			End date of record
DIM_PARTY_POLICHOLDER	EXTERNAL_REFERENCE	varchar(255)	NO		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	ROLEPLAYER_ID	numeric(19,0)	NO		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	ROLEPLAYER_VERSION	numeric(19,0)	NO		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	ROLEPLAYERTYPE_ID	int	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	BIRTH_DATE	datetime	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	ISBIRTHDETIALSESTIMATED	char(1)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	DISPOSABLE_INCOME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	DISPOSABLE_CURRENCYCODE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	GROSS_INCOME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	GROSS_INCOME_CURRENCY_CODE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	EMPLOYMENTSTATUS	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	GENDER	varchar(255)	YES		IAA	PARTY	no	

Table 7-60 Source to Target for Dim Party Policyholder

DIM_PARTY_POLICHOLDER	LANGUAGE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	MARITAL_STATUS	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	INITIALS	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	BIRTH_NAME_ID	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	FIRST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	LAST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	MIDDLE_NAMES	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	PREFIX_TITLES	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	SUFFIX_TITLES	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	SHORTFIRSTNAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	SALUTATION	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	FULLNAME	varchar(1279)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	DESCRIPTION	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	BIRTH_NAME_START_DATE	datetime	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	BIRTH_NAME_TYPE_ID	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	BIRTH_NAME_VERSION	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	COUNTRY_CODE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	PASSPORT_NUMBER2	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	PASSPORT_NUMBER	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	NATIONAL_REGISTRATION_ID	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_POLICHOLDER	NATIONAL_REGISTRATION_PARTY_ID	varchar(255)	YES		IAA	PARTY	no	

Table 7-61 Source to Target for Dim Party Policyholder (Continued)

Source to Target Mapping:DIM_PARTY_INTERMEDIARY (Version 1.04)

Table Name	Column name	Data Type	Allow NULL	Target column description	Source System	Source DB / File	Data transform	Notes
DIM_PARTY_INTERMEDIARY	PARTY_INTERMEDIARY_ID	int IDENTITY(1,1)	NO		new			gets created for each new entry
DIM_PARTY_INTERMEDIARY	SOURCE_SYSTEM_ID	int	YES		DW			Id of source system
DIM_PARTY_INTERMEDIARY	SOURCE_SYSTEM_REFERENCE	varchar(255)	YES		DW			Reference on source system
DIM_PARTY_INTERMEDIARY	SOURCE_SYSTEM_START_DT	datetime	YES		DW			Record start date of source system
DIM_PARTY_INTERMEDIARY	SOURCE_SYSTEM_END_DT	datetime	YES		DW			Record end date of source system
DIM_PARTY_INTERMEDIARY	SOURCE_SYSTEM_STATUS	varchar(9)	YES		DW			Indicator of source system status
DIM_PARTY_INTERMEDIARY	IS_CURRENT	bit	YES		DW			Indicator for current record
DIM_PARTY_INTERMEDIARY	PREVIOUS_ID	int	YES		DW			Points to previous record
DIM_PARTY_INTERMEDIARY	RECORD_START_DT	datetime	YES		DW			Record start date
DIM_PARTY_INTERMEDIARY	RECORD_END_DT	datetime	YES		DW			End date of record
DIM_PARTY_INTERMEDIARY	TITLE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	FIRST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	MIDDLE_NAMES	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	LAST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	KIND	varchar(32)	YES		IAA	PARTY	no	Type of intermediary
DIM_PARTY_INTERMEDIARY	KIND_ID	int	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	FSP	varchar(128)	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	FSP_ID	bigint	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	REPORTS_TO	varchar(64)	YES		IAA	PARTY	no	Reports to consultant
DIM_PARTY_INTERMEDIARY	REPORTS_TO_ID	bigint	YES		IAA	PARTY	no	
DIM_PARTY_INTERMEDIARY	BELONGS_TO	varchar(64)	YES		IAA	PARTY	no	Belongs to area
DIM_PARTY_INTERMEDIARY	BELONGS_TO_ID	bigint	YES		IAA	PARTY	no	

Table 7-62 Source to Target for Dim Party Intermediary

Source to Target Mapping:DIM_PARTY_CONSULTANT (Version 1.04)

Table Name	Column name	Data Type	Allow NULL	Target column description	Source System	Source DB / File	Data transform	Notes
DIM_PARTY_CONSULTANT	PARTY_CONSULTANT_ID	int IDENTITY(1,1)	NO		new			gets created for each new entry
DIM_PARTY_CONSULTANT	SOURCE_SYSTEM_ID	int	YES		DW			Id of source system
DIM_PARTY_CONSULTANT	SOURCE_SYSTEM_REFERENCE	varchar(255)	YES		DW			Reference on source system
DIM_PARTY_CONSULTANT	SOURCE_SYSTEM_START_DT	datetime	YES		DW			Record start date of source system
DIM_PARTY_CONSULTANT	SOURCE_SYSTEM_END_DT	datetime	YES		DW			Record end date of source system
DIM_PARTY_CONSULTANT	SOURCE_SYSTEM_STATUS	varchar(9)	YES		DW			Indicator of source system status
DIM_PARTY_CONSULTANT	IS_CURRENT	bit	YES		DW			Indicator for current record
DIM_PARTY_CONSULTANT	PREVIOUS_ID	int	YES		DW			Points to previous record
DIM_PARTY_CONSULTANT	RECORD_START_DT	datetime	YES		DW			Record start date
DIM_PARTY_CONSULTANT	RECORD_END_DT	datetime	YES		DW			End date of record
DIM_PARTY_CONSULTANT	TITLE	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_CONSULTANT	FIRST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_CONSULTANT	MIDDLE_NAMES	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_CONSULTANT	LAST_NAME	varchar(255)	YES		IAA	PARTY	no	
DIM_PARTY_CONSULTANT	REPORTS_TO	varchar(64)	YES		IAA	PARTY	no	Reports to area manager
DIM_PARTY_CONSULTANT	REPORTS_TO_ID	bigint	YES		IAA	PARTY	no	
DIM_PARTY_CONSULTANT	BELONGS_TO	varchar(64)	YES		IAA	PARTY	no	Belongs to area
DIM_PARTY_CONSULTANT	BELONGS_TO_ID	bigint	YES		IAA	PARTY	no	

Table 7-63 Source to Target for Dim Party Consultant

Source to Target Mapping:DIM_ACTUAL (Version 1.02)

Table Name	Column name	Data Type	Allow NULL	Target column description	Source System	Source DB / File	Data transform	Notes
DIM_ACTUAL	ACTUAL_ID	int IDENTITY(1,1)	NO		new			gets created for each new entry
DIM_ACTUAL	SOURCE_SYSTEM_ID	int	YES		DW			Id of source system
DIM_ACTUAL	SOURCE_SYSTEM_REFERENCE	varchar(255)	YES		DW			Reference on source system
DIM_ACTUAL	SOURCE_SYSTEM_START_DT	datetime	YES		DW			Record start date of source system
DIM_ACTUAL	SOURCE_SYSTEM_END_DT	datetime	YES		DW			Record end date of source system
DIM_ACTUAL	SOURCE_SYSTEM_STATUS	varchar(255)	YES		DW			Indicator of source system status
DIM_ACTUAL	IS_CURRENT	bit	YES		DW			Indicator for current record
DIM_ACTUAL	PREVIOUS_ID	int	YES		DW			Points to previous record
DIM_ACTUAL	RECORD_START_DT	datetime	YES		DW			Record start date
DIM_ACTUAL	RECORD_END_DT	datetime	YES		DW			End date of record
DIM_ACTUAL	MEMBER_EXTERNAL_REFERENCE	varchar(255)	YES		IAA	SPF	no	
DIM_ACTUAL	BUSINESS_GROUP	varchar(255)	YES		IAA	SPF	no	Business area group
DIM_ACTUAL	BENFIT_NAME	varchar(255)	YES		IAA	SPF	no	name of benefit
DIM_ACTUAL	OFFERING	varchar(255)	YES		IAA	SPF	no	name of offering
DIM_ACTUAL	OFFERING_TYPE	varchar(255)	YES		IAA	SPF	no	
DIM_ACTUAL	PRODUCT_TYPE_ID	int	YES		IAA	SPF	no	
DIM_ACTUAL	PRODUCT_DESCRIPTION	varchar(255)	YES		IAA	SPF	no	Product description

Table 7-64 Source to Target for Dim Actual

7.6.3.9. Dimensional modelling design – Develop dimensional tables

Section 7.6.3.8. explains that the development of dimensional tables requires the following analysis documents:

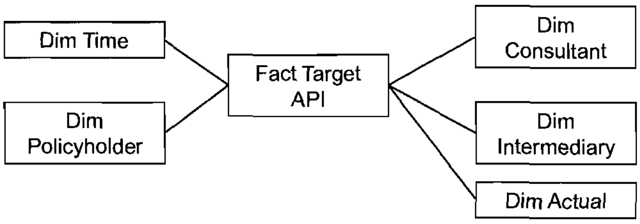
- Data warehouse matrix – illustration of the data marts and the dimensions available for the specific data mart (Figure 7-11).
- Fact table diagram – illustration of the fact table detail within its context (Figure 7-13).
- Dimensional table detail – illustration of the hierarchies in the dimension tables (Table 7-52 to Table 7-57).
- Sources detail – a list of available source data and the owners of the data (Table 7-58).
- Source to target mapping – mapping from the source data to the target dimensional tables (Table 7-59 to Table 7-64).

Figure 7-14 illustrates the star diagrams for the case study.

Star diagram – Fact Target API (version 1.0)

Links to dimensions:

Dim Time
Dim Consultant
Dim Policyholder
Dim Intermediary
Dim Actual



*Refer to source to target mapping for table detail.

Figure 7-14 Star diagram for Fact Target API

Once all the dimensional models are created, the technical architecture model can be created.

7.6.4. Technical Architecture modelling

After completion of the dimensional model, the technical architecture modelling is done. Figure 7-15 illustrates a high-level technical architecture of a typical data warehouse.

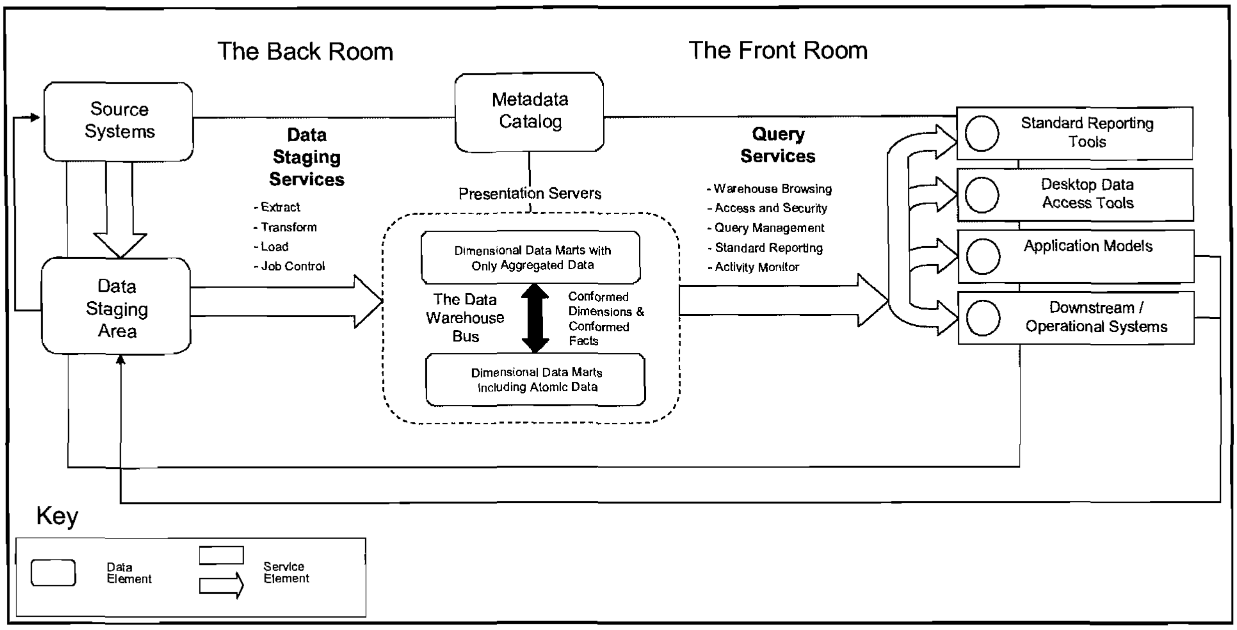


Figure 7-15 High-level technical architecture model (Kimball *et al.*, 1998:329)

Section 6.3.3 explains that the model (Figure 7-15) provides a logical separation between the internal working of the warehouse and the user front end. Therefore, the analysis and design of such architecture should be separated according to the back room and the front room. The analysis and design of the back room entail the following:

- *Source system analysis*
- Data staging services analysis
- Data staging services design

7.6.4.1. Technical Architecture back room OO analysis – Source systems

Section 6.3.3.1 indicates that the analysis needed for the architecture is done during the identification of sources for the dimensional models. The source systems are listed in Table 7-58.

7.6.4.2. Technical Architecture back room OO analysis – Data staging services

Section 6.3.3.2 explains that the data staging services mainly consist of the following:

- Extract
- Transform
- Load
- Job control

Extract

Section 6.3.3.2 explains that the following documentation is needed to analyse the extract design for the data warehouse:

- Source to target mapping for all the dimensions. (Already defined in Table 7-57 to Table 7-64 as part of the discussion)
- Entity relational (ER) model of the source data. (Illustrated in Figure 7-16 to Figure 7-18)
- Business rules that influence the ETL process. (Already defined in Table 7-30 to Table 7-34)

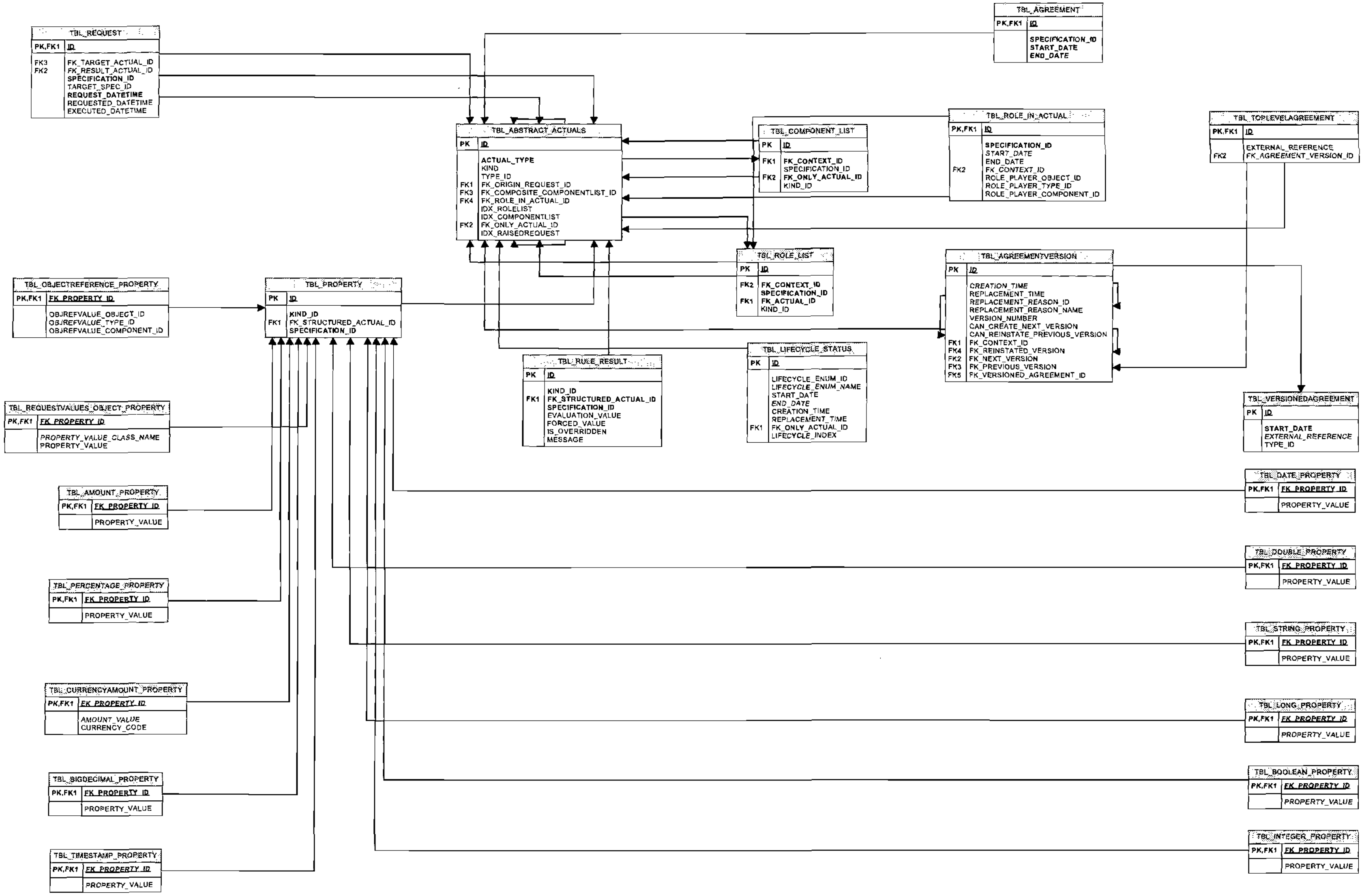


Figure 7-16 ER diagram for IAA-SPF (Part of IAA database)

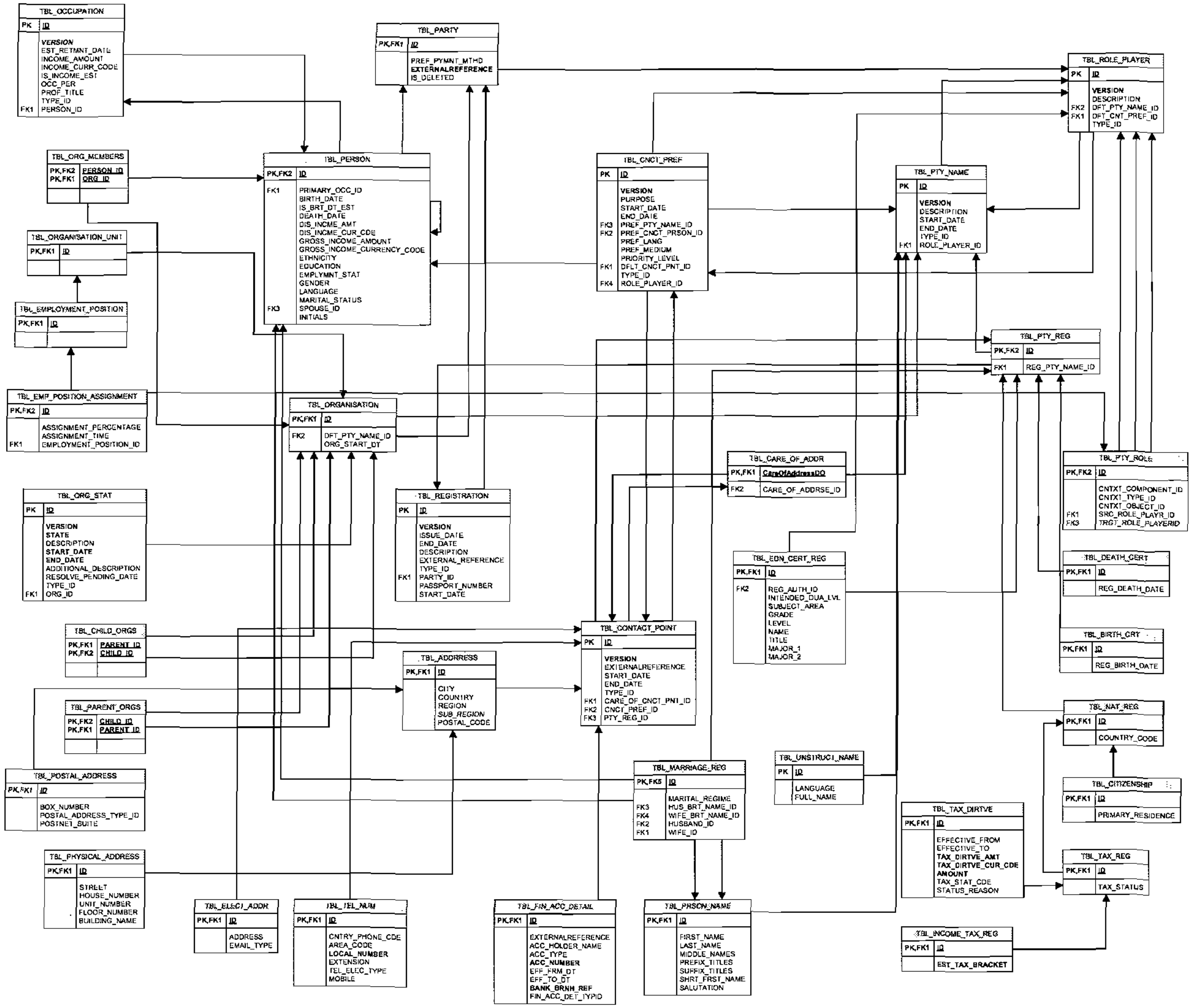


Figure 7-17 ER diagram for IAA-Party (Part of IAA database)

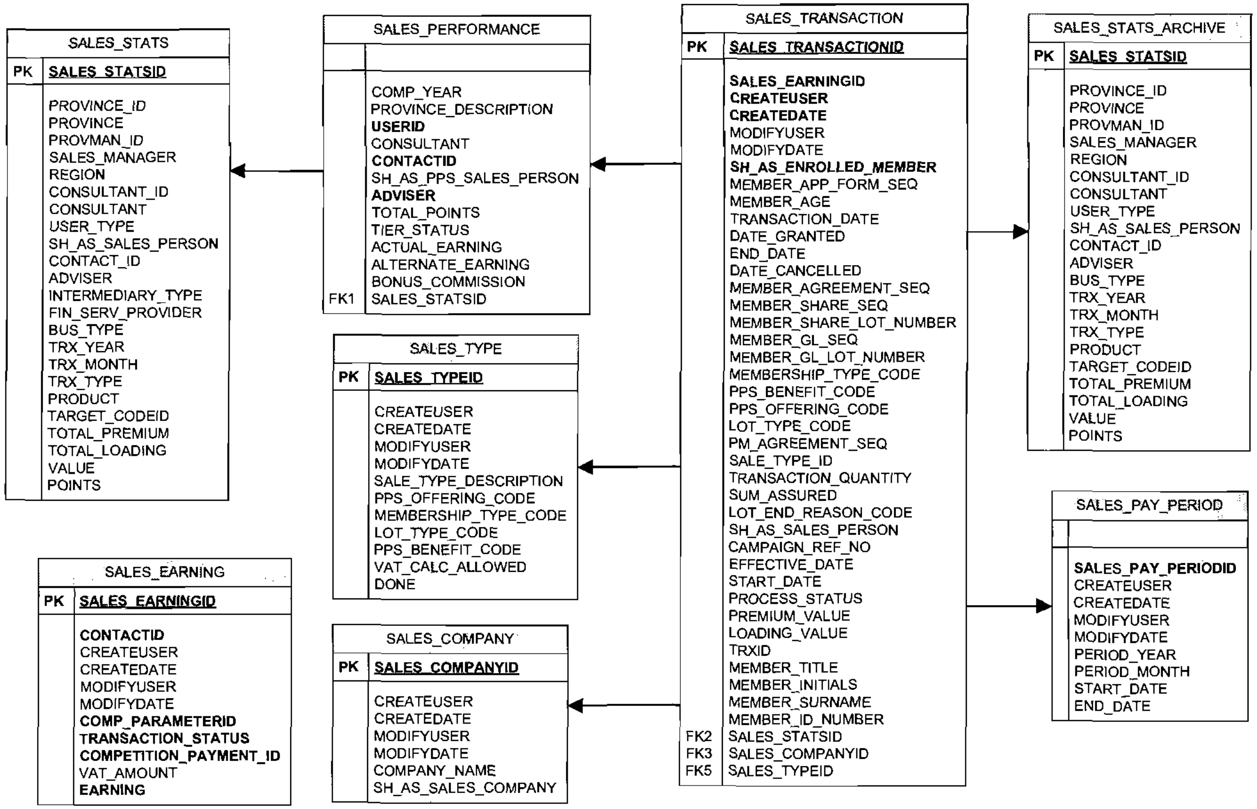


Figure 7-18 ER diagram for SalesLogix

Transform

Transformation requires two types of documents:

- Basic high level data stage schema plan (illustrated in Figure 7-19)
- Detailed plans (illustrated in Figure 7-19 to Figure 7-24)

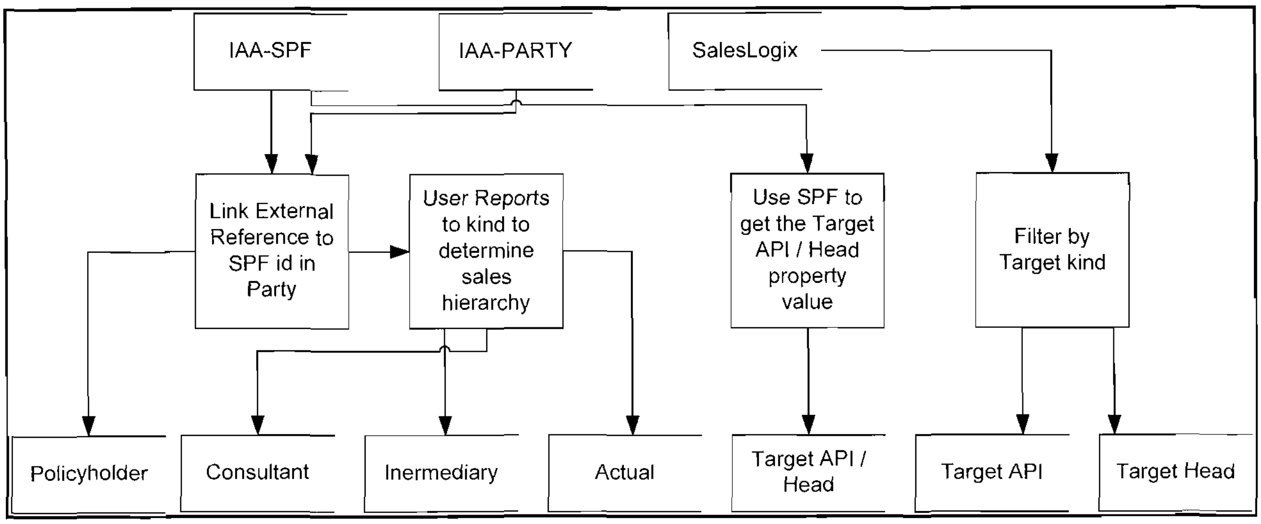

IAA-SPF
IAA-PARTY
SalesLogix
Link External Reference to SPF id in Party
User Reports to kind to determine sales hierarchy
Use SPF to get the Target API / Head property value
Filter by Target kind
Policyholder
Consultant
Inermediary
Actual
Target API / Head
Target API
Target Head

Figure 7-19 High level data stage schema plan for case study

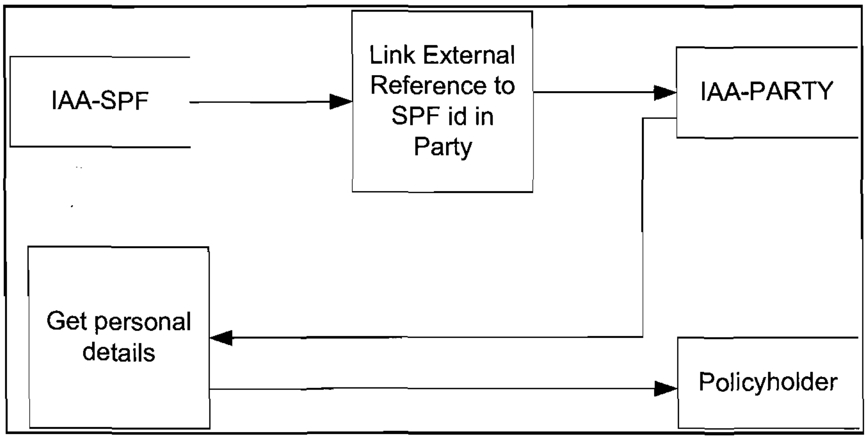


Figure 7-20 Detail level diagram for policyholder dimension extract

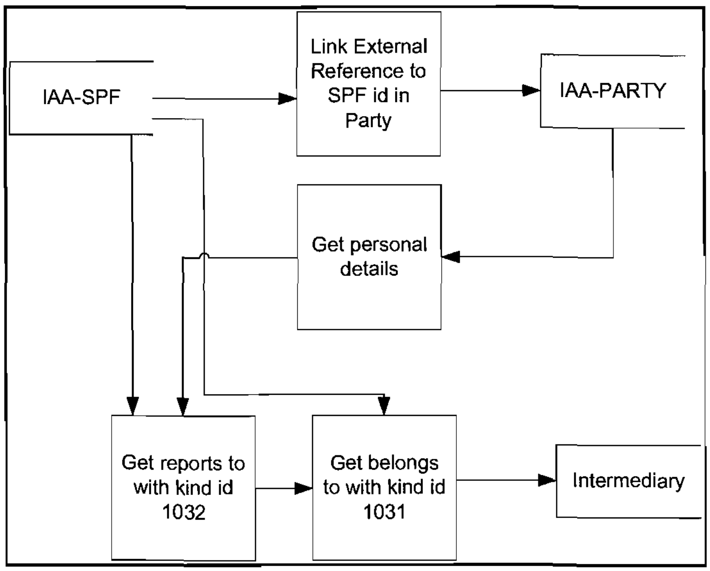

IAA-SPF
Link External Reference to SPF id in Party
IAA-PARTY
Get personal details
Get reports to with kind id 1032
Get belongs to with kind id 1031
Intermediary

Figure 7-21 Detail level diagram for intermediary dimension extract

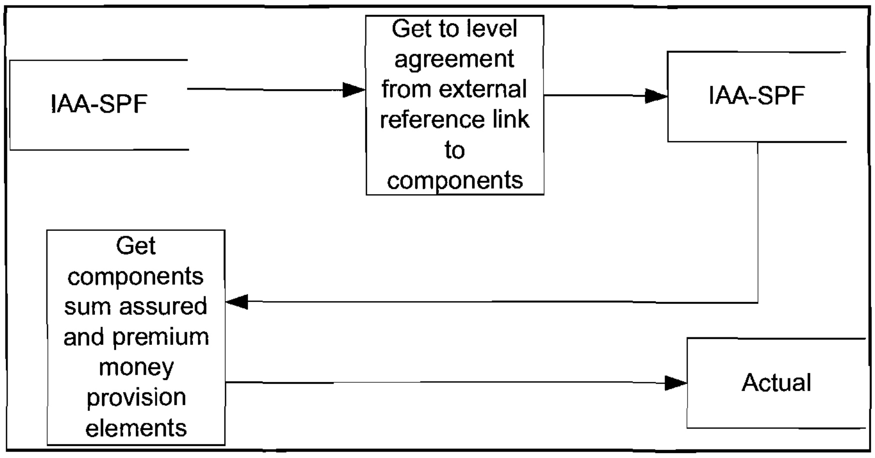

IAA-SPF
Get to level agreement from external reference link to components
IAA-SPF
Get components sum assured and premium money provision elements
Actual

Figure 7-22 Detail level diagram for actual dimension extract

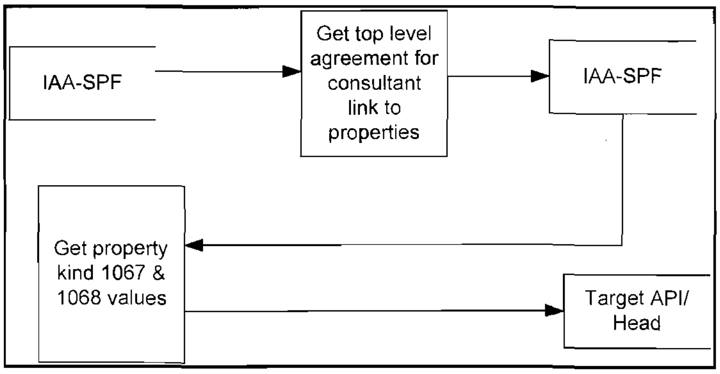

IAA-SPF
Get top level agreement for consultant link to properties
IAA-SPF
Get property kind 1067 & 1068 values
Target API/ Head

Figure 7-23 Detail level diagram for Target API/Head fact extract

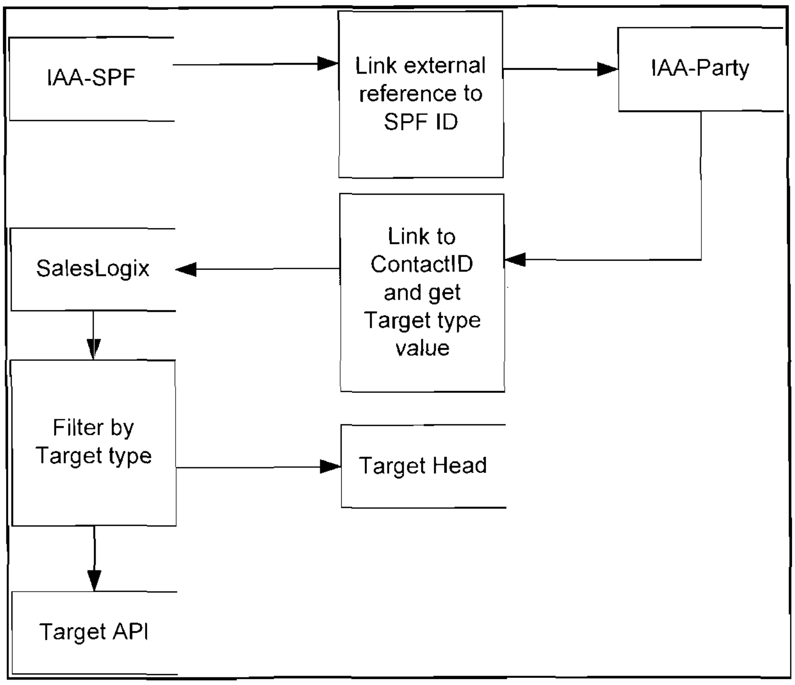

IAA-SPF
Link external reference to SPF ID
IAA-Party
SalesLogix
Link to ContactID and get Target type value
Filter by Target type
Target Head
Target API

Figure 7-24 Detail level diagram for Target API fact and Target Head fact extract

Load

Section 6.3.3.2 explains that the source definition in conjunction with the dimensional table should give the analysis the type of upload it requires. A list should be created listing the sources and the type of upload needed. Table 7-65 illustrates the load type definitions.

Load type definition version 1.0			
Source	**Upload Type**	**Platform**	**Description**
Sales Logix	Transactional	Windows 2003 / SQL Server 2000	Contains all sales data of intermediaries and their consultants.
IAA	Refresh	Windows 2003 / SQL Server 2000	Maintains all party information and policy agreement administration.

Table 7-65 Load type definition for DW case study

Job control

The ETL process needs to be managed, thus the job control services of the ETL. This is done by the deployment team. This team is dedicated toward the following tasks:

- Job scheduling – manage system jobs.
- Monitoring – all databases, application and ETL jobs.
- Logging – faults highlighted are sent to application support.

7.6.4.3. Technical Architecture back room OO design – Data staging services

The analysis documents (gathered for the back room architecture) provide a system independent view of how it should be designed. The design platform used for the data staging service was Microsoft SQL Server 2005 Integration Services (SSIS). This tool was used since all the data bases run on SQL Server 2000 and SQL Server 2005.

7.6.4.4. Technical Architecture front room OO analysis – Query services

The front room is a vital part, as this is the part the users see and use to access the data warehouse (Kimball *et al.*, 1998:409). The user interfaces are in the form of reports, the tool used for this being SQL Reporting Services 2005.

7.6.5. Physical designs

The physical design involves the design of the logical database, as well as its implementation. The process is as follows:

- Define naming standards (Table 7-66)
- Design physical tables and columns (Table 7-68 to Table 7-80)
- Estimate database size and index plan
- Develop aggregation plan

7.6.5.1. Physical Design – Define standards

A document explaining the naming standard for tables, attributes, synonyms and file locations should was created.

Standard definitions (Version 1.1)	
Name	**Description**
SD01	All dimensional tables start with "DIM_".
SD02	All fact tables start with "FACT_".
SD03	All table names are in upper case.
SD04	All varchar data is saved in upper case
SD05	Code page 1562 is used on data bases (default SQL code page)
SD06	All id fields are of data type INT Identity(1,1)
SD07	All SQL Server data files (MDF and LDF files) should be under E:\DATA
SD08	Table attribute names should be in upper case

Table 7-66 Standards definition for the use case

7.6.5.2. Physical Design – Design physical tables and columns

The following tables (Table 7-68 to Table 7-80) provide the physical data layout used for the data warehouse use case.

Physical Data Table Name: IAA_HIERARCHY (Version 1.2)		
Note: Used in promote packages to build the reports to and belong to fields		
Column Name	**Data Type**	**Null Allowed**
SALESDIV_PTY_ID	BIGINT	YES
SALESDIV_PTY_ROLE_ID	BIGINT	YES
SALESDIV_NAME	VARCHAR(64)	YES
SALESDIV_MGR_PTY_ID	BIGINT	YES
SALESDIV_MGR_NAME	VARCHAR(64)	YES
SALESAREA_PTY_ID	BIGINT	YES
SALESAREA_PTY_ROLE_ID	BIGINT	YES
SALESAREA_NAME	VARCHAR(64)	YES
SALESAREA_MGR_PTY_ID	BIGINT	YES
SALESAREA_MGR_NAME	VARCHAR(64)	YES
SALESOFFICE_PTY_ID	BIGINT	YES
SALESOFFICE_PARTOF_ROLE_ID	BIGINT	YES
SALESOFFICE_BELONGSTO_ROLE_ID	BIGINT	YES
SALESOFFICE_NAME	VARCHAR(64)	YES
SALESOFFICE_MGR_SPF_ID	BIGINT	YES
SALESOFFICE_MGR_NAME	VARCHAR(64)	YES
SALESUNIT_PTY_ID	BIGINT	YES
SALESUNIT_NAME	VARCHAR(64)	YES
SALESUNIT_MGR_SPF_ID	BIGINT	YES
SALESUNIT_MGR_NAME	VARCHAR(64)	YES
CONSULTANT_SPF_ID	BIGINT	YES
CONSULTANT_NAME	VARCHAR(64)	YES
CONSULTANT_TITLE	VARCHAR(12)	YES
CONSULTANT_FIRSTNAME	VARCHAR(32)	YES
CONSULTANT_LASTNAME	VARCHAR(32)	YES
CONSULTANT_MIDDLENAMES	VARCHAR(255)	YES
INT_SPF_ID	BIGINT	NO
INT_TITLE	VARCHAR(12)	YES
INT_INITIALS	VARCHAR(12)	YES
INT_FIRSTNAME	VARCHAR(32)	YES
INT_LASTNAME	VARCHAR(32)	YES
INT_FULLNAME	VARCHAR(64)	YES

Table 7-67 Physical table layout for IAA_Hierarchy

INT_EXT_REF	VARCHAR(64)	YES
INT_RECORD_START_DATE	DATETIME	YES
INT_RECORD_END_DATE	DATETIME	YES
LIFECYCLE_ENUM_ID	INT	YES
INT_KIND	INT	YES
INT_KIND_DESCR	VARCHAR(32)	YES
FSP_SPF_ID	BIGINT	YES
FSP_NAME	VARCHAR(128)	YES
LAST_EXTRACT_DATE	DATETIME	YES

Table 7-68 (Continued) Physical table layout for IAA_Hierarchy

Physical Data Table Name: FACT_CONSULTANT_TARGET_PROPERTIES (Version 1.2)		
Note: Used for fact consultant target properties		
Column Name	**Data Type**	**Null Allowed**
ID	INT IDENTITY	NO
DIM_DATE	VARCHAR(8)	YES
PARTY_AREA_MANAGER_ID	INT	YES
PARTY_CONSULTANT_ID	INT	YES
PARTY_AREA_ID	INT	YES
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_VERSION	INT	YES
EXTRACT_DATE	DATETIME	YES
API_FACTOR	DECIMAL(10,2)	YES
API_PERCENTAGE	DECIMAL(10,2)	YES
API_TARGET	DECIMAL(10,2)	YES
COMMISSION_FACTOR	DECIMAL(10,2)	YES
INCENTIVE_VARIABLE_PORTION_OF_SALARY	DECIMAL(10,2)	YES
INTERMEDIARY_FACTOR	DECIMAL(10,2)	YES
INTERMEDIARY_START_DT	DECIMAL(10,2)	YES
NEW_HEAD_FACTOR	DECIMAL(10,2)	YES
NEW_HEAD_PERCENTAGE	DECIMAL(10,2)	YES
NEW_HEAD_TARGET	DECIMAL(10,2)	YES
REGIONAL_FACTOR	DECIMAL(10,2)	YES
TARGET_API_AMOUNT	DECIMAL(10,2)	YES
TARGET_NEW_HEAD_AMOUNT	DECIMAL(10,2)	YES

Table 7-69 Physical table layout for FACT_CONSULTANT_TARGET_PROPERTIES

Physical Data Table Name: FACT_API (Version 1.2)		
Note: Used for fact API		
Column Name	**Data Type**	**Null Allowed**
ID	INT IDENTITY	NO
PARTY_INTERMEDIARY_ID	INT	YES
PARTY_CONSULTANT_ID	INT	YES
DIM_DATE	VARCHAR(8)	YES
PARTY_AREA_MANAGER_ID	INT	YES
ACTUAL_ID	INT	YES
PARTY_POLICYHOLDER_ID	INT	YES
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_EXTRACT_DATE	DATETIME	YES
EXTRACT_DATE	DATETIME	YES
TRANSACTION_AMOUNT	NUMERIC(18,2)	YES
ALLOCATED_AMOUNT	NUMERIC(18,2)	YES
MEMBER_AGE	INT	YES
COMMISSION_TYPE	VARCHAR(255)	YES
MONTHS_IN_FORCE	INT	YES
ORIGINAL_REQUEST_DATE	DATETIME	YES
REQUESTED_DATETIME	DATETIME	YES
TRANSACTION_DATE	DATETIME	YES

Table 7-70 Physical table layout for FACT_HEAD

Physical Data Table Name: DIM_PARTY_POLICYHOLDER (Version 1.2)		
Note: Used for policyholder dimension		
Column Name	**Data Type**	**Null Allowed**
PARTY_POLICYHOLDER_ID	INT IDENTITY	NO
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_REFERENCE	VARCHAR(255)	YES
SOURCE_SYSTEM_START_DT	DATETIME	YES
SOURCE_SYSTEM_END_DT	DATETIME	YES
SOURCE_SYSTEM_STATUS	VARCHAR(255)	YES
IS_CURRENT	BIT	YES
PREVIOUS_ID	INT	YES
RECORD_START_DT	DATETIME	YES
RECORD_END_DT	DATETIME	YES
EXTERNAL_REFERENCE	VARCHAR(255)	NO
ROLEPLAYER_ID	NUMERIC(19,0)	NO
ROLEPLAYER_VERSION	NUMERIC(19,0)	NO
ROLEPLAYERTYPE_ID	INT	YES

Table 7-71 Physical table layout for DIM_PARTY_POLICYHOLDER

BIRTH_DATE	DATETIME	YES
ISBIRTHDETIALSESTIMATED	CHAR(1)	YES
DISPOSABLE_INCOME	VARCHAR(255)	YES
DISPOSABLE_CURRENCYCODE	VARCHAR(255)	YES
GROSS_INCOME	VARCHAR(255)	YES
GROSS_INCOME_CURRENCY_CODE	VARCHAR(255)	YES
EMPLOYMENTSTATUS	VARCHAR(255)	YES
GENDER	VARCHAR(255)	YES
LANGUAGE	VARCHAR(255)	YES
MARITAL_STATUS	VARCHAR(255)	YES
INITIALS	VARCHAR(255)	YES
BIRTH_NAME_ID	VARCHAR(255)	YES
FIRST_NAME	VARCHAR(255)	YES
LAST_NAME	VARCHAR(255)	YES
MIDDLE_NAMES	VARCHAR(255)	YES
PREFIX_TITLES	VARCHAR(255)	YES
SUFFIX_TITLES	VARCHAR(255)	YES
SHORTFIRSTNAME	VARCHAR(255)	YES
SALUTATION	VARCHAR(255)	YES
FULLNAME	VARCHAR(1279)	YES
DESCRIPTION	VARCHAR(255)	YES
BIRTH_NAME_START_DATE	DATETIME	YES
BIRTH_NAME_TYPE_ID	VARCHAR(255)	YES
BIRTH_NAME_VERSION	VARCHAR(255)	YES
COUNTRY_CODE	VARCHAR(255)	YES
PASSPORT_NUMBER2	VARCHAR(255)	YES
PASSPORT_NUMBER	VARCHAR(255)	YES
NATIONAL_REGISTRATION_ID	VARCHAR(255)	YES
NATIONAL_REGISTRATION_PARTY_ID	VARCHAR(255)	YES

Table 7-72 (Continued) Physical table layout for DIM_PARTY_POLICYHOLDER

Physical Data Table Name: DIM_PARTY_INTERMEDIARY (Version 1.2)		
Note: Used for intermediary dimension		
Column Name	**Data Type**	**Null Allowed**
PARTY_INTERMEDIARY_ID	INT IDENTITY	NO
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_REFERENCE	VARCHAR(255)	YES
SOURCE_SYSTEM_START_DT	DATETIME	YES

Table 7-73 Physical table layout for DIM_PARTY_INTERMEDIARY

SOURCE_SYSTEM_END_DT	DATETIME	YES
SOURCE_SYSTEM_STATUS	VARCHAR(9)	YES
IS_CURRENT	BIT	YES
PREVIOUS_ID	INT	YES
RECORD_START_DT	DATETIME	YES
RECORD_END_DT	DATETIME	YES
TITLE	VARCHAR(255)	YES
FIRST_NAME	VARCHAR(255)	YES
MIDDLE_NAMES	VARCHAR(255)	YES
LAST_NAME	VARCHAR(255)	YES
KIND	VARCHAR(32)	YES
KIND_ID	INT	YES
FSP	VARCHAR(128)	YES
FSP_ID	BIGINT	YES
REPORTS_TO	VARCHAR(64)	YES
REPORTS_TO_ID	BIGINT	YES
BELONGS_TO	VARCHAR(64)	YES
BELONGS_TO_ID	BIGINT	YES

Table 7-74 (Continued) Physical table layout for DIM_PARTY_INTERMEDIARY

Physical Data Table Name: DIM_PARTY_CONSULTANT (Version 1.2)		
Note: Used for the consultant dimension		
Column Name	**Data Type**	**Null Allowed**
PARTY_CONSULTANT_ID	INT IDENTITY	NO
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_REFERENCE	VARCHAR(255)	YES
SOURCE_SYSTEM_START_DT	DATETIME	YES
SOURCE_SYSTEM_END_DT	DATETIME	YES
SOURCE_SYSTEM_STATUS	VARCHAR(9)	YES
IS_CURRENT	BIT	YES
PREVIOUS_ID	INT	YES
RECORD_START_DT	DATETIME	YES
RECORD_END_DT	DATETIME	YES
TITLE	VARCHAR(255)	YES
FIRST_NAME	VARCHAR(255)	YES
MIDDLE_NAMES	VARCHAR(255)	YES
LAST_NAME	VARCHAR(255)	YES
REPORTS_TO	VARCHAR(64)	YES

Table 7-75 Physical table layout for DIM_PARTY_CONSULTANT

REPORTS_TO_ID	BIGINT	YES
BELONGS_TO	VARCHAR(64)	YES
BELONGS_TO_ID	BIGINT	YES

Table 7-76 (Continued) Physical table layout for DIM_PARTY_CONSULTANT

Physical Data Table Name: DIM_DATE (Version 1.2)		
Note: Used for the date dimension		
Column Name	**Data Type**	**Null Allowed**
DIM_DATE	VARCHAR(8)	NO
YEAR	INT	YES
MONTH	INT	YES
MONTH_NAME	VARCHAR(50)	YES
DAY	INT	YES
DISPLAY_DATE	VARCHAR(10)	YES
DATE	DATETIME	YES
DAY_NAME	VARCHAR(50)	YES
DAY_OF_WEEK	INT	YES
WEEK_OF_YEAR	INT	YES
SEMESTER	INT	YES
QUARTER	INT	YES

Table 7-77 Physical table layout for DIM_DATE

Physical Data Table Name: DIM_ACTUAL (Version 1.2)		
Note: Used for actual dimension (or agreement dimension)		
Column Name	**Data Type**	**Null Allowed**
ACTUAL_ID	INT IDENTITY	NO
SOURCE_SYSTEM_ID	INT	YES
SOURCE_SYSTEM_REFERENCE	VARCHAR(255)	YES
SOURCE_SYSTEM_START_DT	DATETIME	YES
SOURCE_SYSTEM_END_DT	DATETIME	YES
SOURCE_SYSTEM_STATUS	VARCHAR(255)	YES
IS_CURRENT	BIT	YES
PREVIOUS_ID	INT	YES
RECORD_START_DT	DATETIME	YES
RECORD_END_DT	DATETIME	YES
MEMBER_EXTERNAL_REFERENCE	VARCHAR(255)	YES
BUSINESS_GROUP	VARCHAR(255)	YES

Table 7-78 Physical table layout for DIM_ACTUAL

BENFIT_NAME	VARCHAR(255)	YES
OFFERING	VARCHAR(255)	YES
OFFERING_TYPE	VARCHAR(255)	YES
PRODUCT_TYPE_ID	INT	YES
PRODUCT_DESCRIPTION	VARCHAR(255)	YES

Table 7-79 (Continued) Physical table layout for DIM_ACTUAL

Physical Data Table Name: AUDIT_LOG (Version 1.2)		
Note: Used for audit logging purposes		
Column Name	**Data Type**	**Null Allowed**
ID	INT IDENTITY	NO
START_TIME	DATETIME	YES
END_TIME	DATETIME	YES
DURATION	INT	YES
JOB_CODE	VARCHAR(255)	YES
JOB_DESCRIPTION	VARCHAR(255)	YES

Table 7-80 Physical table layout for AUDIT_LOG

7.6.5.3. Physical Design – Estimate database size and index plan

Estimating the database size was done with the assistance of the DBA in the deployment team. The initial sizes were gathered from the database management system and the growth was estimated by comparing the table sizes after each dimension and fact table promotion. Table 7-81 illustrates the tables in the DW with the sizes and row counts for each table. No aggregation plan was created, as there was no need for this.

Database size and Index plan for DW

Table Name	Row Count	Table Size (KB)	Data Space Used (KB)	Index Space Used (KB)	Unused Space (KB)	Growth Expected in rows per ETL	Table growth (KB)	Table Size per Month (KB)	Growth with
AUDIT_LOG	18	24	16	8	0	18	24	744	Each promotion of the dimensions and fact tables.
DIM_ACTUAL	245136	46800	37656	9096	48	5000	955	29592	Each new policy contract
DIM_DATE	146099	12808	12696	80	32	0	0	0	Static
DIM_PARTY_CONSULTANT	88	72	16	16	40	50	41	1268	Each new consultant
DIM_PARTY_INTERMEDIARY	7938	1800	1760	16	24	500	113	3515	Each new intermediary
DIM_PARTY_POLICYHOLDER	81537	28112	26216	1816	80	2500	862	26720	New policyholder
FACT_API	480583	61320	61032	208	80	2500	319	9889	Each new policy sold
FACT_CONSULTANT_TARGET_PROPERTIES	88	72	16	16	40	100.00	82	2536	Each target set on consultant
FACT_HEAD	144019	10808	10696	80	32	2000	150	4653	New policyholder
IAA_HIERARCHY	31077	13576	13560	8	8	1000	437	13542	Any change to intermediary or consultant

Table 7-81 Database size and index plan for the case study DW

7.6.6. Data staging

Section 6.3.5 furthermore suggests that the main object-oriented phases be investigated. These are:

- Gather requirements
- Analyse requirements
- Design
- Implement
- Test

Gather requirements

Requirements were gathered at the beginning of the project (section 7.6.2.) while analysis took place during dimensional modelling (section 7.6.3.) and technical architecture modelling (section 7.6.4.).

Analyse requirements

The analysis of the backroom architecture serves as the planning part for the ten step overview (steps 1–4). The analysis of the backroom provides the developers with a one page schematic flow, the strategy of the data stage tool to be implemented and a detailed schematic of the data restructuring and transformation process. In terms of object-oriented phases, the dimension loads (step 4 – 6), as well as fact table and automation (steps 7 – 10) consist of a design phase, an implementation phase and a testing phase. The rest of section 7.6.6. will deal with data staging for the case study DW.

7.6.6.1. Data Staging – OO Design

The data staging environment has two major design areas:

- Dimension table loading
- Fact table loading and automation

The analysis documents available for the designs are:

- Dimension model designs (section 7.6.3.9.)
- Entity Relationship models of the sources systems (section 7.6.3.8.)
- High level schematic plan (section 7.6.4.3.)
- Detail schematic plans (section 7.6.4.3.)
- Business rules (section 7.6.5.1.)

Section 6.3.5.1 explains that based on the analysis documents listed, the following need to be created for each dimensional load and fact table load in the data staging area:

- State chart model
- Entity relationship model
- Collaboration model between ETL processes

State chart model

The state chart model should illustrate extraction of the data from its starting point (the source) to the transformation and its conditions to its end point (the dimensions or fact tables). Figure 7-25 to Figure 7-30 illustrate the state chart diagrams used for the data staging logic.

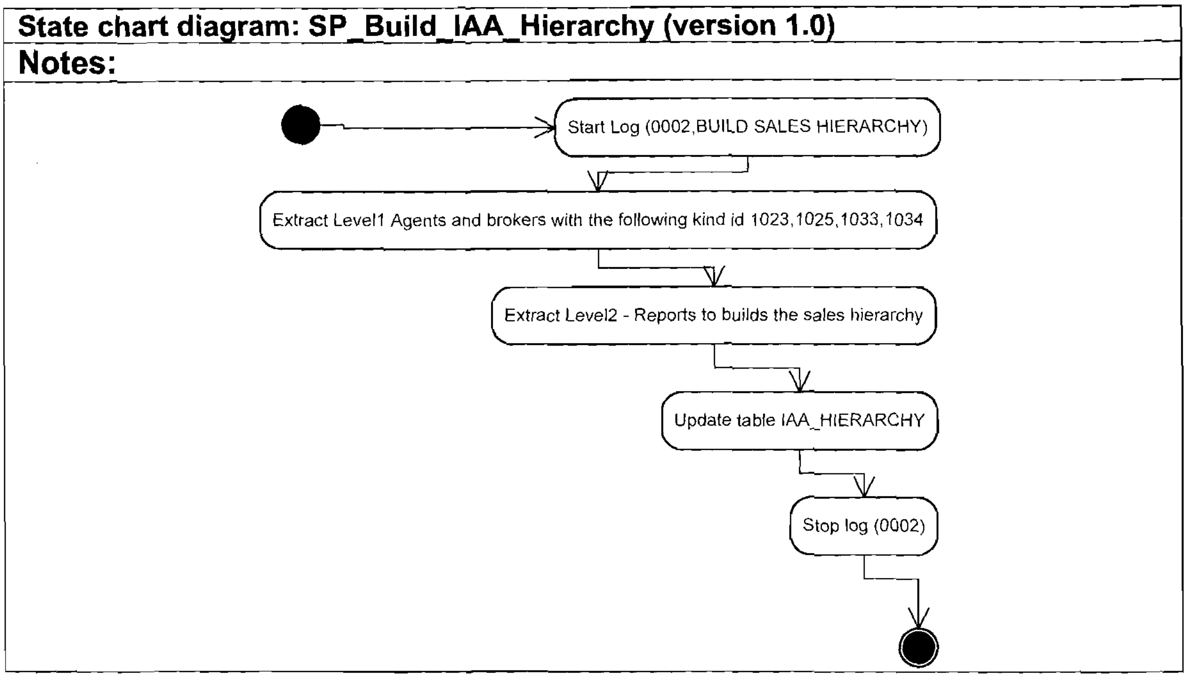

State chart diagram: SP_Build_IAA_Hierarchy (version 1.0)
Notes:
Start Log (0002,BUILD SALES HIERARCHY)
Extract Level1 Agents and brokers with the following kind id 1023,1025,1033,1034
Extract Level2 - Reports to builds the sales hierarchy
Update table IAA_HIERARCHY
Stop log (0002)

Figure 7-25 State chart model for SP_BUILD_IAA_HIERARCHY

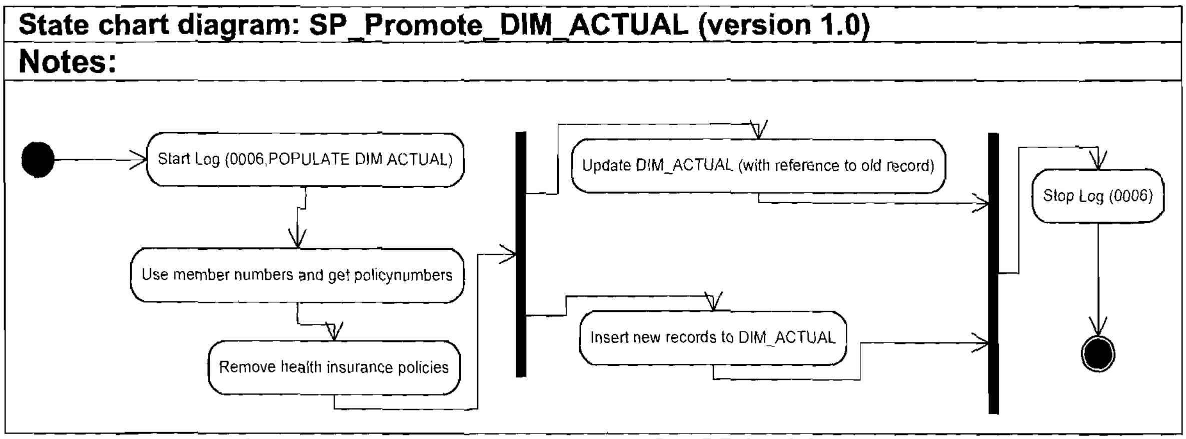

State chart diagram: SP_Promote_DIM_ACTUAL (version 1.0)
Notes:
Start Log (0006,POPULATE DIM ACTUAL)
Use member numbers and get policynumbers
Remove health insurance policies
Update DIM_ACTUAL (with reference to old record)
Insert new records to DIM_ACTUAL
Stop Log (0006)

Figure 7-26 State chart model for SP_PROMOTE_ACTUAL

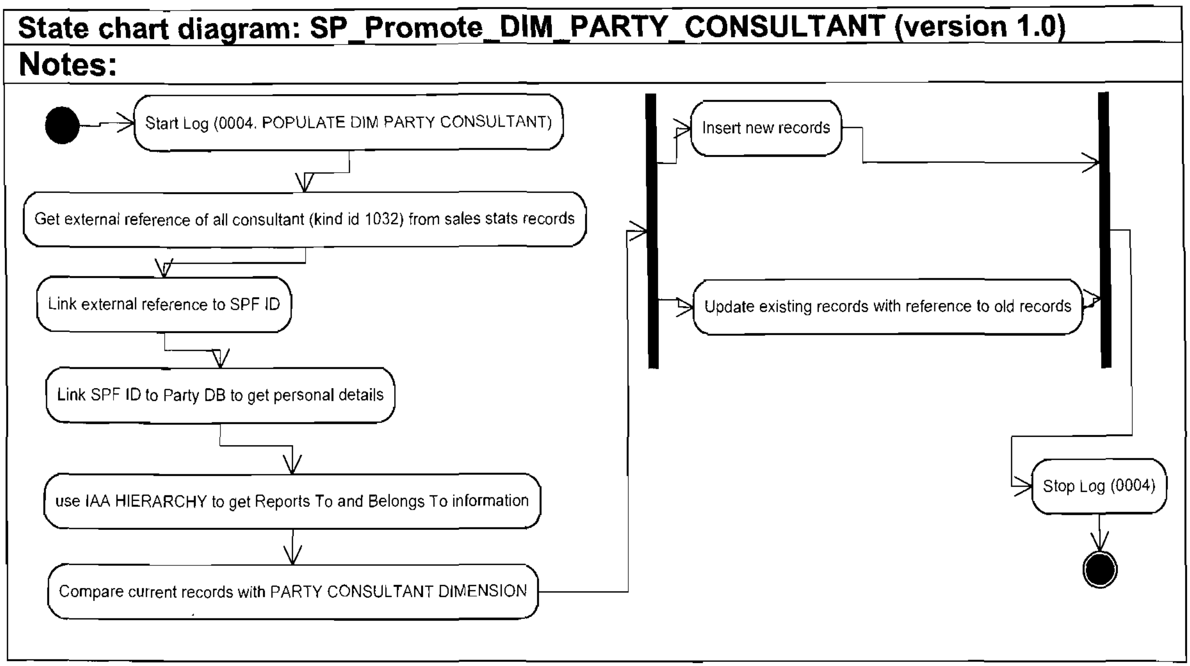

State chart diagram: SP_Promote_DIM_PARTY_CONSULTANT (version 1.0)
Notes:
Start Log (0004. POPULATE DIM PARTY CONSULTANT)
Get external reference of all consultant (kind id 1032) from sales stats records
Link external reference to SPF ID
Link SPF ID to Party DB to get personal details
use IAA HIERARCHY to get Reports To and Belongs To information
Compare current records with PARTY CONSULTANT DIMENSION
Insert new records
Update existing records with reference to old records
Stop Log (0004)

Figure 7-27 State chart model for SP_PROMOTE_PARTY_CONSULTANT

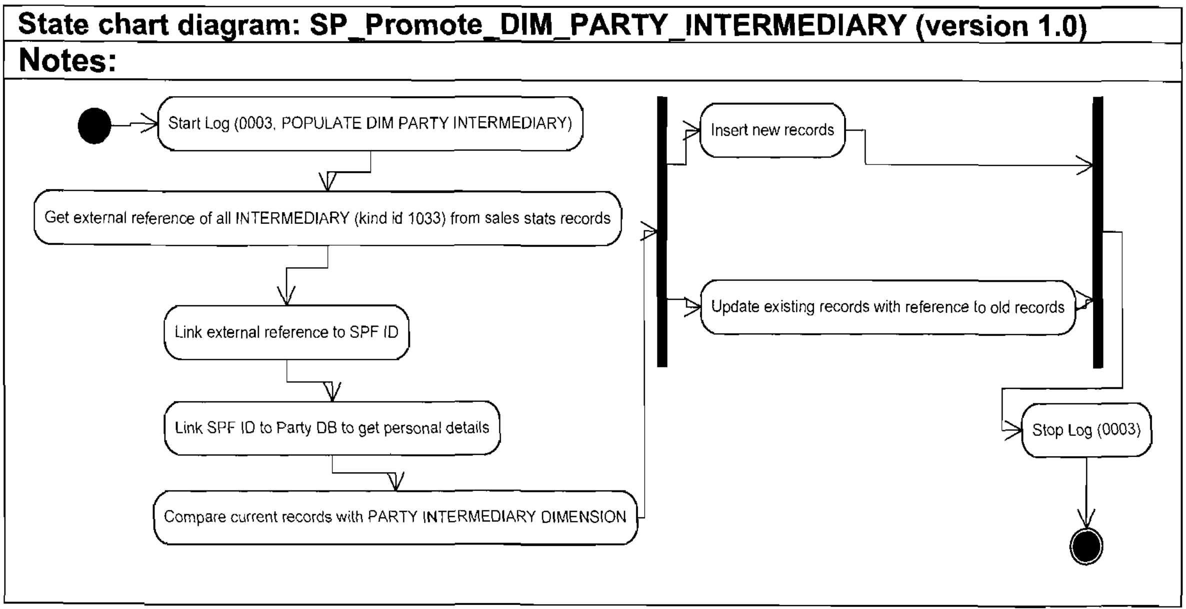

State chart diagram: SP_Promote_DIM_PARTY_INTERMEDIARY (version 1.0)
Notes:
Start Log (0003. POPULATE DIM PARTY INTERMEDIARY)
Get external reference of all INTERMEDIARY (kind id 1033) from sales stats records
Link external reference to SPF ID
Link SPF ID to Party DB to get personal details
Compare current records with PARTY INTERMEDIARY DIMENSION
Insert new records
Update existing records with reference to old records
Stop Log (0003)

Figure 7-28 State chart model for SP_PROMOTE_PARTY_INTERMEDIARY

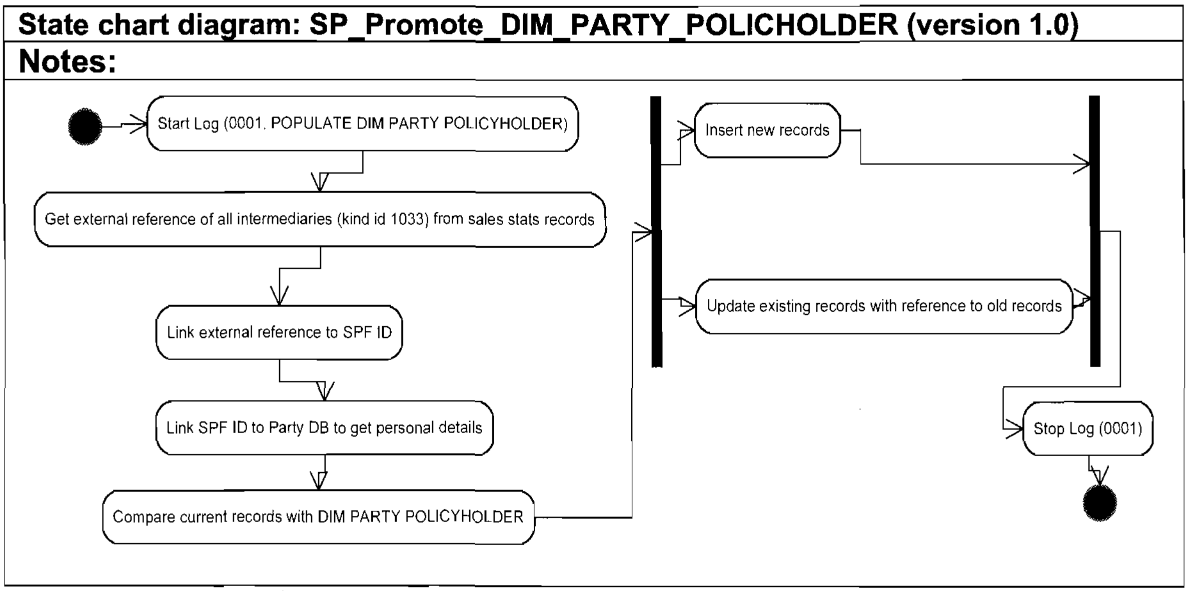

State chart diagram: SP_Promote_DIM_PARTY_POLICHOLDER (version 1.0)
Notes:
Start Log (0001. POPULATE DIM PARTY POLICYHOLDER)
Get external reference of all intermediaries (kind id 1033) from sales stats records
Link external reference to SPF ID
Link SPF ID to Party DB to get personal details
Compare current records with DIM PARTY POLICYHOLDER
Insert new records
Update existing records with reference to old records
Stop Log (0001)

Figure 7-29 State chart model for SP_PROMOTE_PARTY_POLICYHOLDER

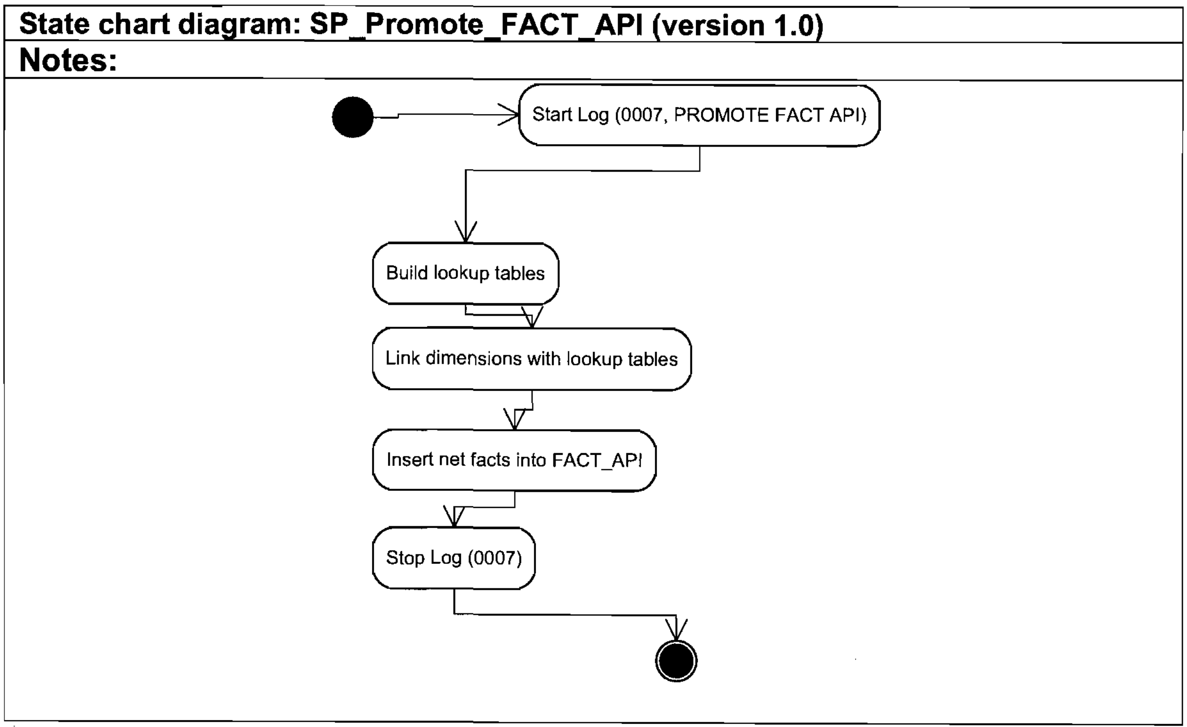

State chart diagram: SP_Promote_FACT_API (version 1.0)
Notes:
Start Log (0007, PROMOTE FACT API)
Build lookup tables
Link dimensions with lookup tables
Insert net facts into FACT_API
Stop Log (0007)

Figure 7-30 State chart model for SP_PROMOTE_FACT_API

Entity relationship model

Accompanied by the state chart models, is an entity relationship model illustrating the underlying structure that will support the ETL. Figure 7-31 shows the ER model supporting the ERL processes illustrated in Figure 7-25 to Figure 7-30.

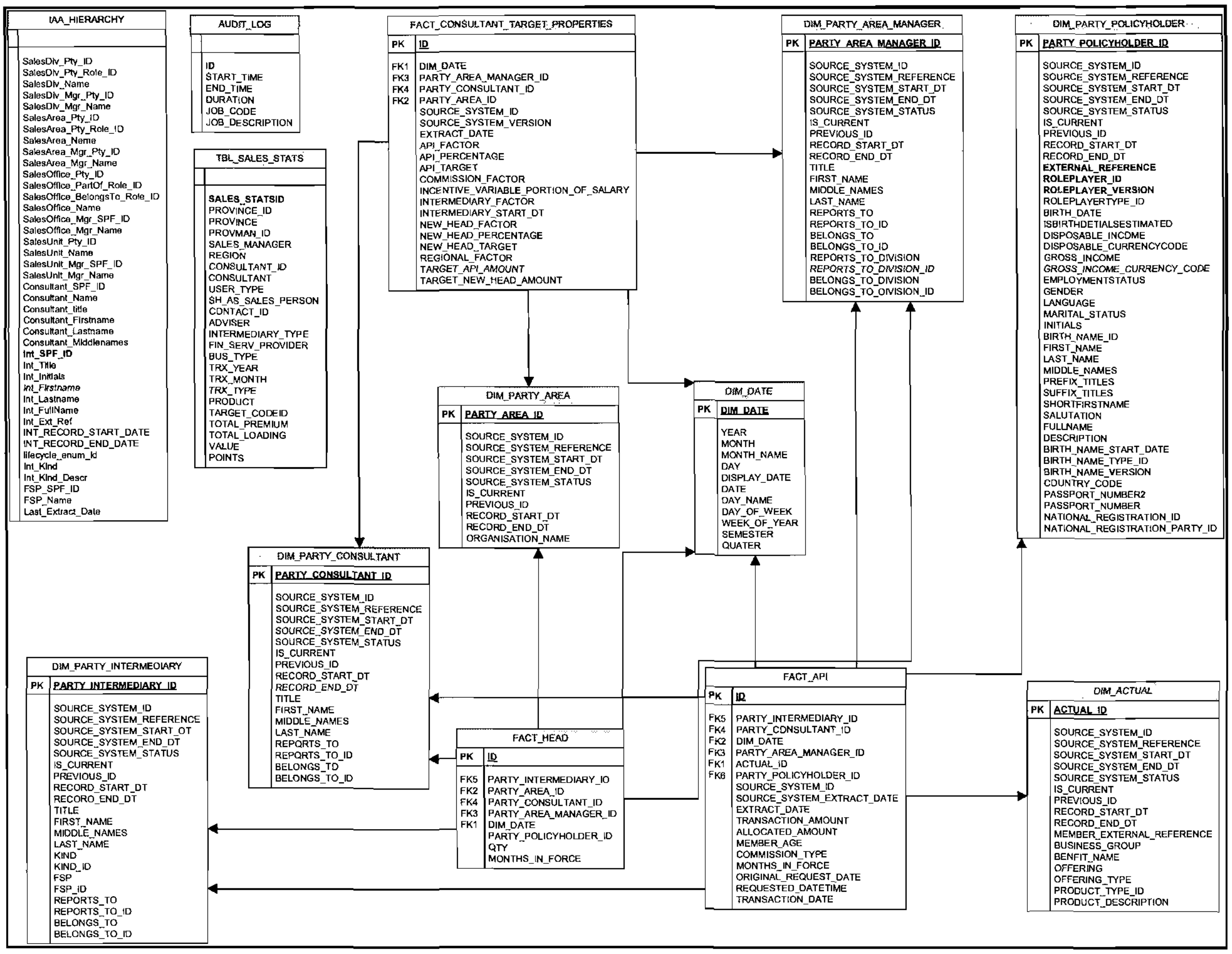


Figure 7-31 ER Model for staging environment for sales ETL

Collaboration model

The collaboration model is created once all the state chart models and entity relationship models are done. The collaboration model provides a graphical overview of the interaction between the ETL processes and their inter-dependencies. Figure 7-32 shows the collaboration diagram for the ETL process in the DW case study.

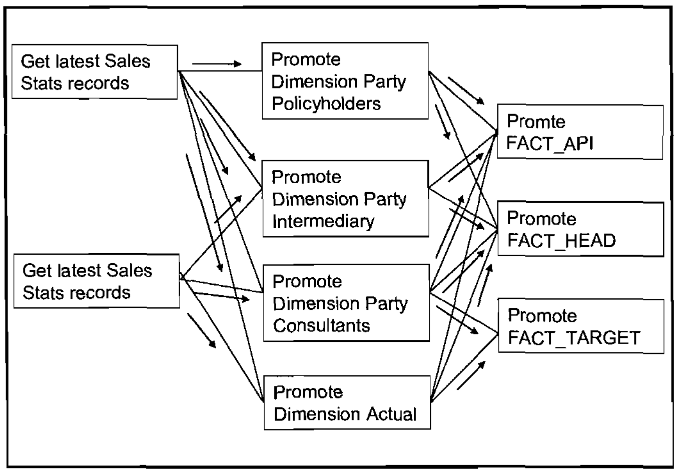
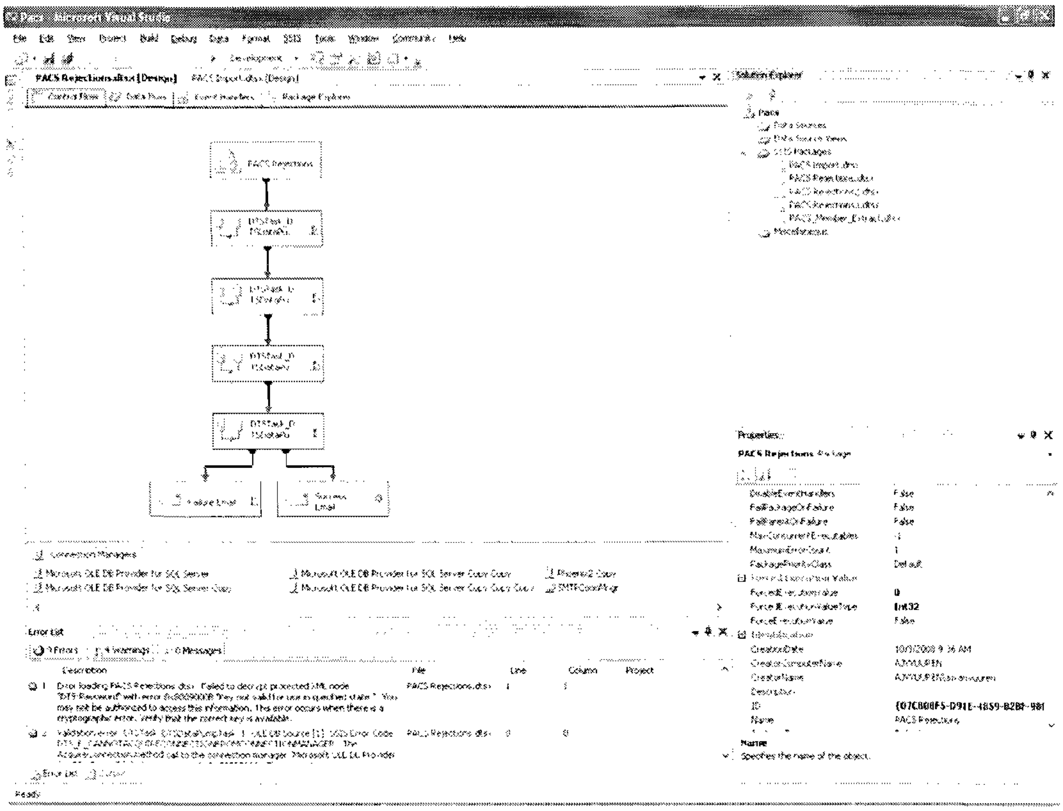

Get latest Sales Stats records
Promote Dimension Party Policyholders
Promte FACT_API
Promote Dimension Party Intermediary
Promote FACT_HEAD
Get latest Sales Stats records
Promote Dimension Party Consultants
Promote FACT_TARGET
Promote Dimension Actual

Figure 7-32 Collaboration diagram on the ETL for the data warehouse

7.6.6.2. Data Staging – OO Implementation

The implementation of the dimension and fact table loads is supported by the state chart and ER model designs created. Each stored procedure is grouped as a package and coded. The packages are stored on the SQL server itself as stored procedures.

The flow control of these packages is controlled by another stored procedure that kicks off the ETL stored procedure according to the collaboration diagram illustrated in Figure 7-33.



PACS - Microsoft Visual Studio
File Edit View Project Build Debug Data Format SSIS Tools Window Community Help
PACS Rejections.dtsx [Design]
PACS Import.dtsx [Design]
Control Flow
Data Flow
Event Handlers
Package Explorer
Solution Explorer
PACS
Data Sources
Data Source Views
SSIS Packages
PACS Import.dtsx
PACS Rejections.dtsx
PACS_Member_Extract.dtsx
Miscellaneous
PACS Rejections
Success Email
Connection Managers
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Provider for SQL Server Copy
Microsoft OLE DB Provider for SQL Server Copy Copy
SMTPConnMgr
Error List
Output
Ready

Properties

PACS Rejections Package

DisableEventHandlers	False
FailPackageOnFailure	False
FailParentOnFailure	False
MaxConcurrentExecutables	-1
MaximumErrorCount	1
PackagePriorityClass	Default
ForcedExecutionValue	
ForcedExecutionValueType	Int32
ForceExecutionValue	False
Identification	
CreationDate	10/3/2008 9:36 AM
CreatorComputerName	[illegible]
CreatorName	[illegible]
Description	
ID	{07C800F5-D91E-4859-B2B8-98I
Name	PACS Rejections

Name
Specifies the name of the object.

2 Errors | 4 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	Error loading PACS Rejections.dtsx: Failed to decrypt protected XML node "DTS:Password" with error 0x8009000B "Key not valid for use in specified state." You may not be authorized to access this information. This error occurs when there is a cryptographic error. Verify that the correct key is available.	PACS Rejections.dtsx	1	1	
2	Validation error. [illegible] OLE DB Source [1]: SSIS Error Code DTS_E_CANNOTACQUIRECONNECTIONFROMCONNECTIONMANAGER. The AcquireConnection method call to the connection manager "Microsoft OLE DB Provider [illegible]	PACS Rejections.dtsx	0	0	

Figure 7-33 SSIS that controls the flow of the stored procedures

7.6.6.3. Data Staging – OO Testing

Unit testing was done on each package by means of the following:

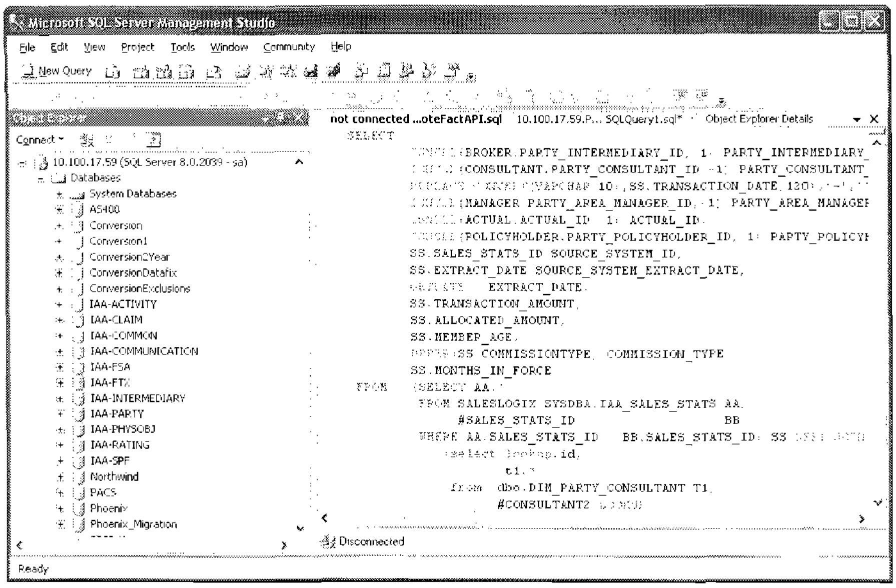
- Duplication testing
- Reconciliation testing, by comparing figures on reports per time frame to production report figures.
- Changing dimension testing, by changing key fields in the dimension and investigating whether a new record is created.

7.6.7. End user applications

The end user application for the case study must support the following groups:

- Application support (IT)
- Business (Sales Department)

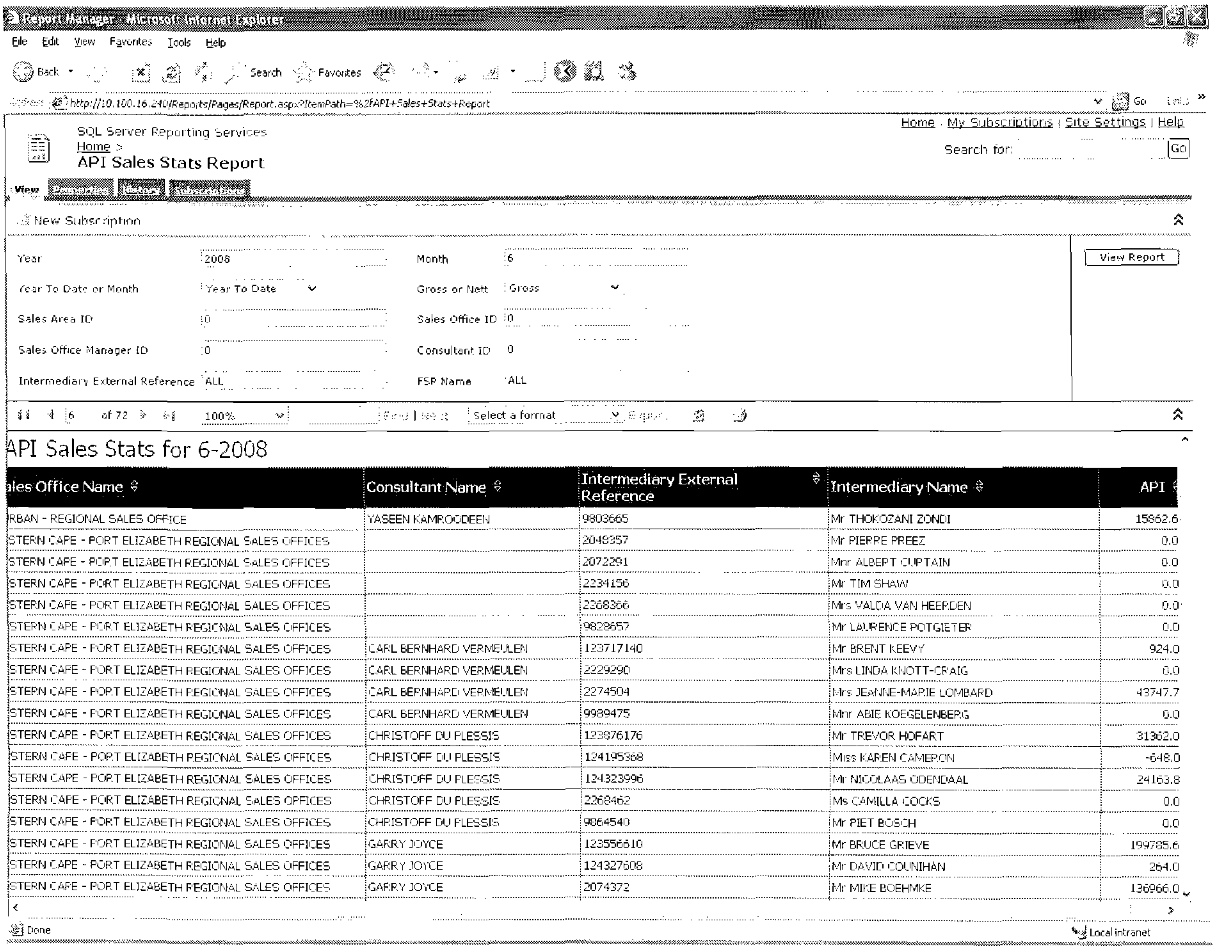
Application support means the creation of *ad hoc* queries as requested by various managers in different departments. Microsoft SQL Management Studio was used to create and run these queries. *Ad hoc* queries have to go through a staging phase, followed by QA and finally production for execution. The results of the production execution are sent to the business. Figure 7-34 is an illustration of the tool used by developers to create *ad hoc* reports.



```
SELECT
        [illegible](BROKER.PARTY_INTERMEDIARY_ID, 1) PARTY_INTERMEDIARY_
        [illegible](CONSULTANT.PARTY_CONSULTANT_ID -1) PARTY_CONSULTANT_
        [illegible](VARCHAR(10),SS.TRANSACTION_DATE,120)[illegible]
        [illegible](MANAGER.PARTY_AREA_MANAGER_ID,-1) PARTY_AREA_MANAGER
        [illegible](ACTUAL.ACTUAL_ID  1) ACTUAL_ID,
        [illegible](POLICYHOLDER.PARTY_POLICYHOLDER_ID, 1) PARTY_POLICYH
        SS.SALES_STATS_ID SOURCE_SYSTEM_ID,
        SS.EXTRACT_DATE SOURCE_SYSTEM_EXTRACT_DATE,
        [illegible]   EXTRACT_DATE,
        SS.TRANSACTION_AMOUNT,
        SS.ALLOCATED_AMOUNT,
        SS.MEMBER_AGE,
        [illegible](SS.COMMISSIONTYPE, COMMISSION_TYPE
        SS.MONTHS_IN_FORCE
    FROM    (SELECT AA.*
         FROM SALESLOGIX.SYSDBA.IAA_SALES_STATS AA,
              #SALES_STATS_ID                        BB
         WHERE AA.SALES_STATS_ID  BB.SALES_STATS_ID) SS [illegible]
            (select lookup.id,
                    t1.*
             from  dbo.DIM_PARTY_CONSULTANT T1,
                    #CONSULTANT2 [illegible]
```

Figure 7-34 SQL Server Manager Studio used by developers

Regular reports as requested, were created using Microsoft SQL Reporting Services. The report also needs to go though the development life cycle of staging, QA and production. Once the report is in production, the end user may visit a web site on the intranet of the company and request the report. Figure 7-35 is an example of an end user report in the case study.


Report Manager - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
http://10.100.16.240/Reports/Pages/Report.aspx?ItemPath=%2fAPI+Sales+Stats+Report
Go
Home My Subscriptions Site Settings Help
Search for: Go
View Properties History Subscriptions
New Subscription
View Report
6 of 72 100% Select a format
Done
Local intranet

SQL Server Reporting Services
Home >
API Sales Stats Report

Year	2008	Month	6
Year To Date or Month	Year To Date	Gross or Nett	Gross
Sales Area ID	0	Sales Office ID	0
Sales Office Manager ID	0	Consultant ID	0
Intermediary External Reference	ALL	FSP Name	ALL

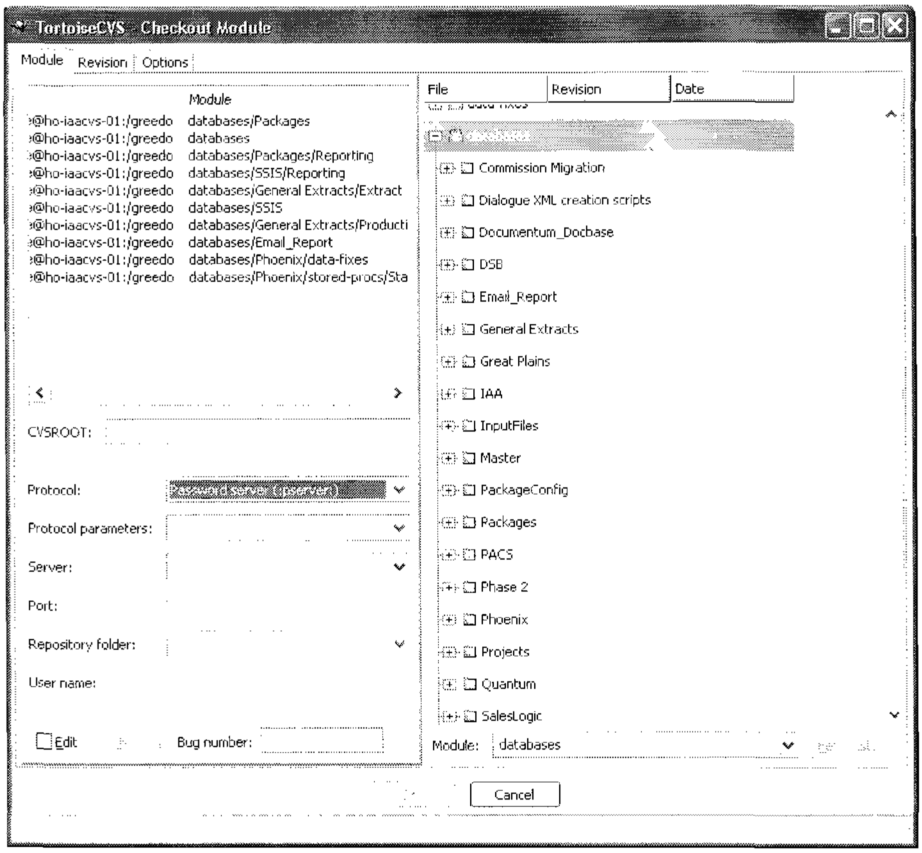
API Sales Stats for 6-2008

ales Office Name	Consultant Name	Intermediary External Reference	Intermediary Name	API
RBAN - REGIONAL SALES OFFICE	YASEEN KAMROODEEN	9803665	Mr THOKOZANI ZONDI	15862.6
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES		2048357	Mr PIERRE PREEZ	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES		2072291	Mnr ALBERT CURTAIN	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES		2234156	Mr TIM SHAW	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES		2268366	Mrs VALDA VAN HEERDEN	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES		9828657	Mr LAURENCE POTGIETER	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CARL BERNHARD VERMEULEN	123717140	Mr BRENT KEEVY	924.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CARL BERNHARD VERMEULEN	2229290	Mrs LINDA KNOTT-CRAIG	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CARL BERNHARD VERMEULEN	2274504	Mrs JEANNE-MARIE LOMBARD	43747.7
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CARL BERNHARD VERMEULEN	9989475	Mnr ABIE KOEGELENBERG	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CHRISTOFF DU PLESSIS	123876176	Mr TREVOR HOFART	31362.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CHRISTOFF DU PLESSIS	124195368	Miss KAREN CAMERON	-648.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CHRISTOFF DU PLESSIS	124323996	Mr NICOLAAS ODENDAAL	24163.8
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CHRISTOFF DU PLESSIS	2268462	Ms CAMILLA COCKS	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	CHRISTOFF DU PLESSIS	9864540	Mr PIET BOSCH	0.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	GARRY JOYCE	123556610	Mr BRUCE GRIEVE	199785.6
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	GARRY JOYCE	124327608	Mr DAVID COUNIHAN	264.0
STERN CAPE - PORT ELIZABETH REGIONAL SALES OFFICES	GARRY JOYCE	2074372	Mr MIKE BOEHMKE	136966.0

Figure 7-35 End user app report

7.6.8. Deployment

All end user applications should go through a development stage, i.e. staging, QA and production. The source files are typically stored on a repository system called CVS. This repository system supports different streams of development. A developer needs to checkout code before developing. Once the changes are done, the developer can check in the code. Figure 7-36 illustrates a CVS's checkout module used to get the latest copy of source code on the developer's PC.



	Module
:@ho-iaacvs-01:/greedo	databases/Packages
:@ho-iaacvs-01:/greedo	databases
:@ho-iaacvs-01:/greedo	databases/Packages/Reporting
:@ho-iaacvs-01:/greedo	databases/SSIS/Reporting
:@ho-iaacvs-01:/greedo	databases/General Extracts/Extract
:@ho-iaacvs-01:/greedo	databases/SSIS
:@ho-iaacvs-01:/greedo	databases/General Extracts/Producti
:@ho-iaacvs-01:/greedo	databases/Email_Report
:@ho-iaacvs-01:/greedo	databases/Phoenix/data-fixes
:@ho-iaacvs-01:/greedo	databases/Phoenix/stored-procs/Sta

CVSROOT:

Protocol: Password server (:pserver:)

Protocol parameters:

Server:

Port:

Repository folder:

User name:

□ Edit Bug number:

File	Revision	Date

- Commission Migration
- Dialogue XML creation scripts
- Documentum_Docbase
- DSB
- Email_Report
- General Extracts
- Great Plains
- IAA
- InputFiles
- Master
- PackageConfig
- Packages
- PACS
- Phase 2
- Phoenix
- Projects
- Quantum
- SalesLogic

Module: databases

Figure 7-36 CVS as repository system for source code

In this case study scripts are used as a mechanism for packaging code. The code developed, is tested on a staging server. Once the developer is satisfied with the code, he/she checks this into CVS. A request is logged with the deployment team to promote the new code to QA. An appointed person in the relevant business department then signs off the QA code in QA. The deployment team promotes all signed off changes twice a month from QA to production. Dates are scheduled when changes are promoted.

7.6.9. Maintenance and growth

The deployment team is responsible for scheduling the required ETL jobs for the DW. Any defects in the DW or its end user application are reported to application support's service desk. A service desk call is logged and assigned to the responsible person. The code changes need go through the development life cycle discussed earlier.

All enhancements to the DW are driven by a project manager and plan. All plans are treated as new development projects.

Currently, there is a need for more information by departments other than the sales department. Each of these requirements will be treated as a future development phase.

7.7. Summary

This chapter deals with the research design and implementation of one of the development methodologies discussed in chapter 6. In the research design, the research question was to determine whether a data warehouse can be developed using a data warehouse methodology and incorporating object-oriented techniques. To answer the above research question, it was necessary to implement one of the methodologies discussed in chapter 6. This chapter describes the implementation of OO and the Business dimensional lifecycle approach to DW development.

An insurance company serves as the research environment requiring multiple parties for the development of the DW. Based on the methodology implementation experience, it was found that the following techniques worked very well in an OO development culture:

- Essential Use case diagram
- User interface prototyping
- Domain modelling
- Supplementary documentation

Although the concept of a traditional essential use case was altered in chapter 6, it was still easily understood by business analysts. The rest of the requirements gathering techniques followed a business and data warehouse structure. Owing to the nature of the technical environment, as well as analysis and design of the data marts (described from section 7.6.3.4.), one was forced to repeatedly reassess the designs of the data marts. In some cases, reassessment of the requirements was necessary to accommodate the design. One such case was the change in membership types of the new production system (IAA) to allow for temporary members and members only. This is evidence of an iterative approach common to OO development.

The following techniques from the business dimensional lifecycle are used in implementation:

- Data warehouse bus architecture
- Fact table diagram
- Dimension table detail
- Source identification

Based on the above analysis techniques, the fact and dimensional tables were developed.

The back room architecture of the technical architecture model involved analysis of the following:

- Source systems
- Data staging services

The data staging services were based on the source systems identified, as well as the designs for the star diagrams.

The front room analysis followed an OO approach on its own, as the reporting tool used is based on the Microsoft.Net architecture, which is an OO framework.

The physical design phase in the business dimensional lifecycle approach was implemented using a design, implementation and testing phase. The design involves the design of the data warehouse, as well as the design of the data staging area.

The design of the data warehouse entails the following:

- Defining standards
- Designing physical tables and columns
- Estimating database size and index plan
- Designing aggregates – no aggregation tables were defined

The design of the data staging area entails the following:

- Designing a state chart diagram based on the analysis of the data staging services in the technical architecture
- Designing an ER model supporting the state chart model
- Designing a collaboration model depicting the overall ETL schema

The implementation of the data warehouse and the data staging involved the coding of the designs created.

The testing of the data staging phase uses the same approach than the OO approach. The testing of ETL jobs was done in phases suggesting that it is unit tested. Audit statistics is a technique used to test the data staging.

The end user application supports the front end of the technical architecture design, and a complete OO model can be applied to the development of these applications.

Chapter 8 - Evaluation of the IS Prototype

8.1. Introduction

Baskerville (1999:13) explains that part of the approach to action research is to evaluate and specify what is learned from the action plan implemented. Therefore, this chapter will deal with the evaluation and specification of knowledge gained from the implementation of the data warehouse (DW) described in chapter 6.

The aim of the evaluation is to determine whether the techniques used to implement the DW in the insurance company were successful. The evaluation was done by investigating the incident requests and change requests logged at the application support service desk of the business. Interviews were also conducted with the business analyst and the application support manager to obtain their perception of the techniques used.

The evaluation process is summarised, followed by a discussion on specified learning based on the feedback received. The specified learning discussion focuses on each object-oriented (OO) phase, detailing each technique in the phase used.

8.2. The research question and the evaluation

This section deals with the evaluation of the DW implemented. It forms part of the evaluation phase of action research as explained in Baskerville (1999:13) and illustrated in Figure 8-1.

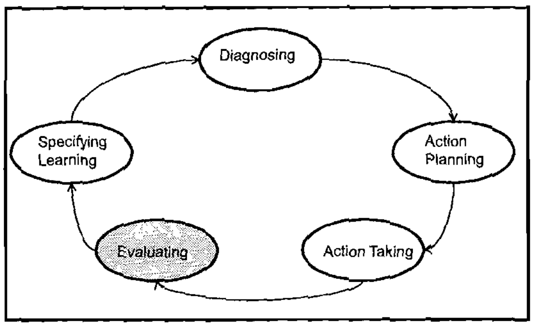


Figure 8-1 Evaluation in the Action Research Cycle (Baskerville, 1999:14)

The previous chapter dealt with the development of a DW, using a DW methodology and jointly incorporating object-oriented (OO) techniques.

To measure the success of the above-mentioned implemented DW, the following key areas need to be defined:

- User acceptance testing
- Incident requests from application support
- Change requests from application support
- Evaluation of OO techniques and DW methods

8.2.1. User acceptance testing

The sponsor of the DW project came from the application support department of the insurance company. User acceptance testing (UAT) is done by the manager and assigned testers of that department.
UAT testing involves the following:

- Evaluating reports generated from the DW
- Evaluating *ad hoc* SQL results and comparing these to existing results in the current production environment.

Normal evaluation is done by means of test packs. A test pack is a set of tests developed by the testers to evaluate the functionality of the application. No test packs were available for the DW, as the latter was introduced to the company.

Test packs will be developed at a later stage. Currently, testing is done by comparing results from the DW in the quality assurance (QA) environment to those from the DW in the production environment, or in some cases the reporting system in the production environment. The QA environment is an environment similar to the production environment but, used for evaluation purposes only.

Each release of the DW is referred to as a build. A build is released to QA where testers need to sign off or reject the build. A version number is assigned to each build released to QA. This version number corresponds to the build notes and is used to track the changes made to the DW.

Build notes typically contain two types of changes:

- Incident requests
- Change requests

An incident request refers to a defect in the DW build in production and will be discussed in detail in the following section. A change request is an enhancement request to the DW build in production. A discussion on this follows in section 8.2.3.

8.2.2. Incident requests

One of the controls in the application support department is incident requests. This request is logged by a user, if he/she experiences problems with a specific application (in this case the DW implemented). The request is forwarded to the appropriate business analyst, who will evaluate the problem and pass it on to the developer responsible. From here the developer applies the necessary changes to correct the error and submit them to the deployment process where the required user acceptance testing is done.

The following is a list of general incidents logged at the service desk:

- Report on API did not display the correct API count. This was applicable to any time period
- Head count report did not display correct head count on certain consultants
- Claw back commission report displayed incorrect amounts

The following is a discussion on the use of OO techniques for the DW and the above logged incident requests.

8.2.2.1. Incident requests and traditional DW methodology

This section explores the possibility of whether the incidents (listed in the previous section) result from using OO techniques to develop the DW, or not.

The first common incident is a report on API that did not display the correct API count. This was valid for any time period. The cause of the incident was due to a party type not loaded in the intermediary dimension. The initial requirements only highlighted one "broker" type, with the second broker type, "agent", never catered for. The above incident was resolved by including the missing broker type "agent" in the extract transform and loading (ETL). The root cause of the above incident was due to a missing kind id (referring to the "agent" type) in the state chart diagram document for the promote sp_promote_DIM_PARTY_INTERMEDIARY ETL. The requirements collection was done correctly, as the physical data table DIM_PARTY_INTERMEDIARY has a kind and kind_id field. This indicates that the design of the above ETL was faulty. As the requirements and the analysis were done correctly, one cannot state that this type of error is a result of using OO techniques. The chance of getting a similar incident when using another DW methodology would be greater, as requirements collection techniques in OO interrogate the business processes. In this case, the requirements collection was done successfully (as proved by the existence of the fields in the required table), but the design was overseen.

The second common incident was a head count report not reporting the correct head count on certain consultants. The cause of the incident was due to a duplication of policyholder details found in the policyholder dimension table, as the ETL did not cater for scenarios where a policyholder has cancelled his/hers membership and then started a new membership thereafter.

The above incident was resolved by catering for the above scenario in the ETL. The root cause of the above incident was due to incomplete analysis. This possibility was not discussed during the requirements gathering as this was not evident in the requirements documentation. The analysis of the requirements should have explored the possibility of the above member lifecycle behaviour. Owing to the missing information on this type of behaviour in the analysis documentation, the design never catered for it. However, the dimensional tables in the DW needed no change to accommodate the above behaviour. This incident could not result from using OO techniques in DW, as the dimensional tables did make provision for such a scenario.

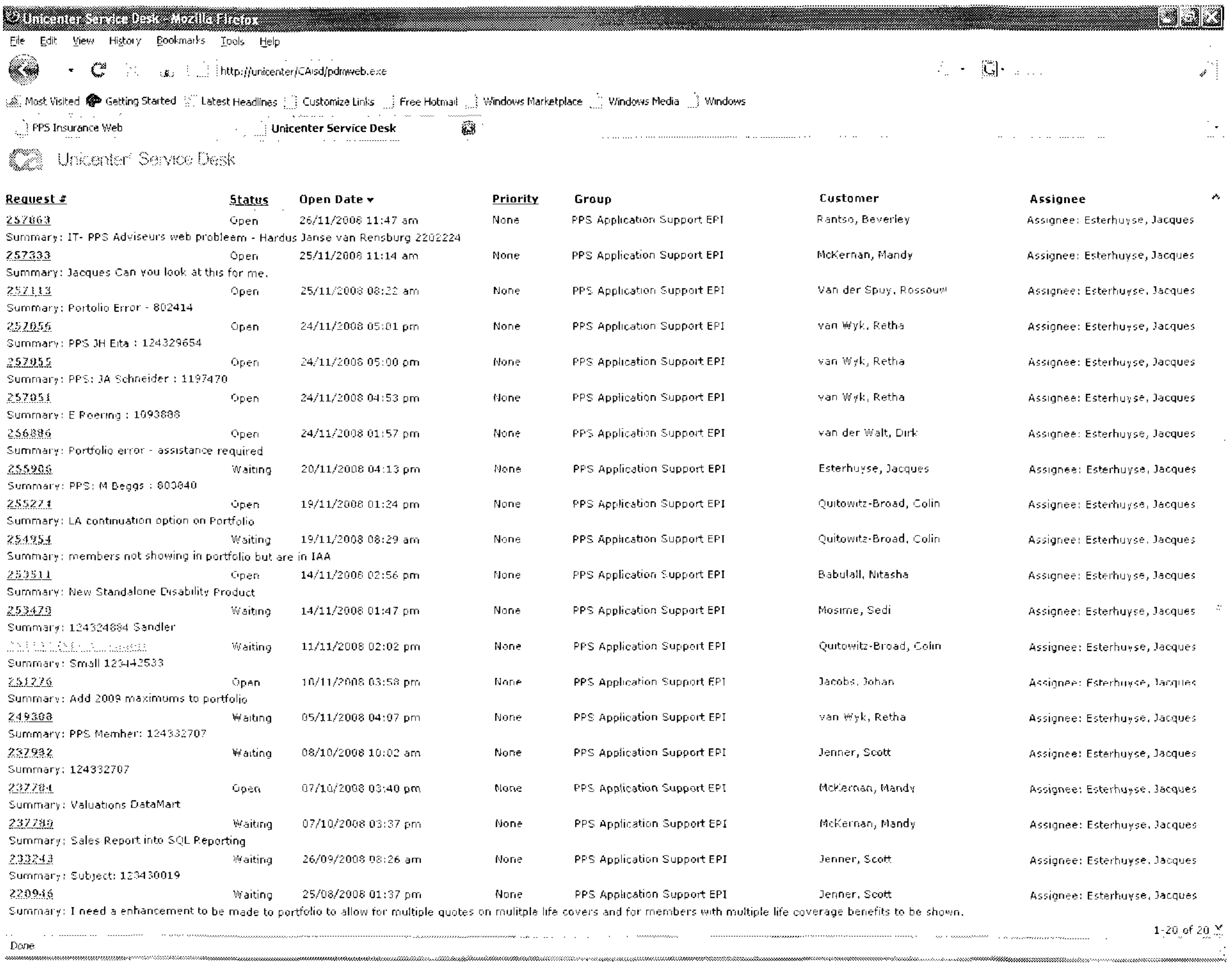
The last common incident was the claw back commission report reporting incorrect amounts. The cause of the incident was missing policyholder details, such as the id number, gender, birth date, or incorrect contact preferences. This was largely due to corruption of data during previous production system migrations. Although controls were set to bar incorrect data, some of it still got promoted to the DW. No changes were needed in the DW presentation layer to accommodate missing data. Most of the changes were made in the ETL for the dimensions. This incident will occur with any DW methodology because of the nature of migrated data. Common migration problems were identified during the analysis of the ETL to cater for scenarios, but not all scenarios are always identifiable. Therefore, this incident did not occur as a result of using OO techniques.

8.2.3. Change requests

A change request is an enhancement request logged by a manager of the requesting department. This request is forwarded to the application support manager who decides whether the change is needed, and if so, assigns it to a business analyst. The business analyst gathers the requirements from the relevant department and supplies the developer with a change request specification document. This document is a signed-off agreement between the application support department and the requesting department for the changes needed.

The only change request was to add a function to the Target API to group it by division. No changes were needed on the DW, as it already contained the division in the dimension consultant (this is referred to as “belongs to”). The changes were needed in the scripts to generate the report.

Figure 8-2 is an illustration of the application used to manage change and incident requests.


Unicenter Service Desk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://unicenter/CAisd/pdmweb.exe
Most Visited Getting Started Latest Headlines Customize Links Free Hotmail Windows Marketplace Windows Media Windows
PPS Insurance Web
Unicenter Service Desk
Unicenter Service Desk
Done
1-20 of 20

Request #	Status	Open Date ▾	Priority	Group	Customer	Assignee
257863	Open	26/11/2008 11:47 am	None	PPS Application Support EPI	Rantso, Beverley	Assignee: Esterhuyse, Jacques
Summary: IT- PPS Adviseurs web probleem - Hardus Janse van Rensburg 2202224						
257333	Open	25/11/2008 11:14 am	None	PPS Application Support EPI	McKernan, Mandy	Assignee: Esterhuyse, Jacques
Summary: Jacques Can you look at this for me.						
257113	Open	25/11/2008 08:22 am	None	PPS Application Support EPI	Van der Spuy, Rossouw	Assignee: Esterhuyse, Jacques
Summary: Portolio Error - 802414						
257056	Open	24/11/2008 05:01 pm	None	PPS Application Support EPI	van Wyk, Retha	Assignee: Esterhuyse, Jacques
Summary: PPS JH Eita : 124329654						
257055	Open	24/11/2008 05:00 pm	None	PPS Application Support EPI	van Wyk, Retha	Assignee: Esterhuyse, Jacques
Summary: PPS: JA Schneider : 1197470						
257051	Open	24/11/2008 04:53 pm	None	PPS Application Support EPI	van Wyk, Retha	Assignee: Esterhuyse, Jacques
Summary: E Poering : 1093888						
256886	Open	24/11/2008 01:57 pm	None	PPS Application Support EPI	van der Walt, Dirk	Assignee: Esterhuyse, Jacques
Summary: Portfolio error - assistance required						
255906	Waiting	20/11/2008 04:13 pm	None	PPS Application Support EPI	Esterhuyse, Jacques	Assignee: Esterhuyse, Jacques
Summary: PPS: M Beggs : 803840						
255274	Open	19/11/2008 01:24 pm	None	PPS Application Support EPI	Quitowitz-Broad, Colin	Assignee: Esterhuyse, Jacques
Summary: LA continuation option on Portfolio						
254954	Waiting	19/11/2008 08:29 am	None	PPS Application Support EPI	Quitowitz-Broad, Colin	Assignee: Esterhuyse, Jacques
Summary: members not showing in portfolio but are in IAA						
253511	Open	14/11/2008 02:56 pm	None	PPS Application Support EPI	Babulall, Nitasha	Assignee: Esterhuyse, Jacques
Summary: New Standalone Disability Product						
253479	Waiting	14/11/2008 01:47 pm	None	PPS Application Support EPI	Mosime, Sedi	Assignee: Esterhuyse, Jacques
Summary: 124324884 Sandler						
[illegible]	Waiting	11/11/2008 02:02 pm	None	PPS Application Support EPI	Quitowitz-Broad, Colin	Assignee: Esterhuyse, Jacques
Summary: Small 123442533						
251276	Open	10/11/2008 03:58 pm	None	PPS Application Support EPI	Jacobs, Johan	Assignee: Esterhuyse, Jacques
Summary: Add 2009 maximums to portfolio						
249308	Waiting	05/11/2008 04:07 pm	None	PPS Application Support EPI	van Wyk, Retha	Assignee: Esterhuyse, Jacques
Summary: PPS Member: 124332707						
237982	Waiting	08/10/2008 10:02 am	None	PPS Application Support EPI	Jenner, Scott	Assignee: Esterhuyse, Jacques
Summary: 124332707						
237784	Open	07/10/2008 03:40 pm	None	PPS Application Support EPI	McKernan, Mandy	Assignee: Esterhuyse, Jacques
Summary: Valuations DataMart						
237780	Waiting	07/10/2008 03:37 pm	None	PPS Application Support EPI	McKernan, Mandy	Assignee: Esterhuyse, Jacques
Summary: Sales Report into SQL Reporting						
233243	Waiting	26/09/2008 08:26 am	None	PPS Application Support EPI	Jenner, Scott	Assignee: Esterhuyse, Jacques
Summary: Subject: 123430019						
220946	Waiting	25/08/2008 01:37 pm	None	PPS Application Support EPI	Jenner, Scott	Assignee: Esterhuyse, Jacques
Summary: I need a enhancement to be made to portfolio to allow for multiple quotes on mulitple life covers and for members with multiple life coverage benefits to be shown.						

Figure 8-2 Application used to manage incidents and change requests

8.2.4. Evaluation of OO and DW

The parties involved in the development of the DW were the following:

- A developer (the researcher)
- A business analyst
- Application support manager

The above mentioned parties had a good understanding of OO development with only the developer understanding DW techniques. Therefore, one of the challenges was to create a DW allowing for individuals with OO background to understand the methodology.

Table 8-2 to Table 8-7 represent an interview with the business analyst and the application support manager to ascertain how they understood the methodology followed in chapter 7.

Question	Business Analyst	Application support manager
	Requirements gathering	
Was the concept of the DW use case diagram used properly?	"The DW use case diagram allows one to define the different areas to focus on when deciding to develop a system. In this case it was successful."	From a business point of view, the DW use case is a very effective way of communicating with the different business units and the people responsible for certain departments. In some sense it also illustrates the collaboration between certain departments and their managers. In this sense, the DW use case diagram did achieve to illustrate the different areas of the business, and compared to the DW use case used in normal development this diagram did not differ.
Was the concept of the DW use cases used properly?	"The DW use cases did differ slightly from what is used normally in use cases. A normal DW use case describes a process in a sequence of steps. This seems to provide information of what data there could be. If the idea of a DW use case is to define what information is available, then it was used successfully." The grain is not a concept used in normal use cases but "defining the so-called grain of the use case is a new concept, but understandable." The reasoning of the grain makes sense, as this seems to be the "driving force" of the process or business department that is being interviewed.	"Defining each input for a department is somewhat unnecessary, as this can become very large and not always useful." "Defining the grain is a way of determining what data is used or generated for the relevant department's job." The use case diagram "highlighted the different areas that one can focus on for developing data marts", and one can easily identify the relation between the DW use case diagram and the DW matrix.

Table 8-1 Requirements gathering on development of the DW using OO techniques

Is the concept of a business process use case diagram useful?	The business process use case diagram “seems like another name for a normal use case diagram.” The idea of a use case diagram is to (as previously mentioned) capture the sequence of steps of a process. “This technique does seem to do just that.”	The business process use case diagram “is an excellent tool to convey what the department does. It is also a very good starting point to determine which reports are needed and what is needed on these reports.”
Was CRC a meaningful technique to determine the problem domain?	“CRC is not a tool that is commonly used in the development culture of this company. Some of the domain objects were defined but there are other ways to do this as well.”	N/A
Was the supplementary documentation meaningful?	Yes it was. “Nothing out of the ordinary.”	N/A
Did you understand the techniques used?	“Mostly yes, as many of these techniques are based on the normal development ones.” The only technique that differed is “the DW use case diagram and the DW use cases though similar to the normal use case”. “Defining the grain for the DW use case is a new concept.”	“Yes, these techniques were successful as a communication tool between colleagues”, “I was able to identify the different business areas and the key stakeholders within the departments.”

Table 8-2 (Continued) Requirements gathering on development of the DW using OO techniques

Question	Business Analyst	Application support manager
Analysis of requirements		
Any comments on the system use cases used?	"This was similar to the essential use cases with only system requirements added." The DW systems use cases "seems to be an unnecessary step, it had only the data type added, the rest was exactly the same." Most of the technical detail could have been added during the essential DW use case.	"It is understandable that the system use cases are done separately from the essential use cases, but one should consider creating a hybrid type of use case to save some time." "Probably the main reason for the feeling of using one use case is because in other projects only one use case is created. The data types are defined in a data extraction definition (DED) document. The DED is accompanied by the use case."
Was the sequence diagram meaningful?	"No not really, the idea of a sequence diagram in general is to describe the follow of a process and the objects between it. This is normally down to a systems level where the objects are defined and the steps are clearly laid out. This sequence diagram seemed to attempt to describe what the link is between that steps and its objects, but the level is too high.	N/A
What is your comments on the data warehouse bus architect matrix	"It does seem that the DW use case with the common objects identified complements the DW matrix." The matrix was not correct the first time, but was completed with an iterative approach between the object identified and the dimensions defined in the matrix. "The matrix does make you think about what information is really available for the DW and how this information corresponds to what needs to be modelled, in other words, should dimension x be available for data mart y."	"This is an excellent way to provide a bigger picture of the project. The matrix helped with defining project scope, modules for the DW and indicates the dependencies between the data marts (departments in the business) and the common concepts in the business, such as a broker or a policyholder, etc." The matrix also serves as a mean to test the correctness of the analysis, for example "to check if there is interaction between a policyholder and the product learning data mart."
Any comments on the dimensional table analysis?	It had to be changed (as used in chapter 7) to be more descriptive and meaningful.	N/A
Any comments on the fact table analysis?	"Not really, was straight forward, the facts were easily identifiable."	N/A

Table 8-3 Analysis phase on development of the DW using OO techniques

Any comments on the hierarchy analysis and design?	"In terms of the business it was clear what needed a hierarchy."	N/A
Any comments on the technical architecture modelling analysis?	N/A	N/A
In terms of OO, were the techniques used meaningful?	"Yes, because the system use cases are definitely used in traditional OO development. The sequence diagrams are not used in the correct way as it should be used. The reason is because sequence diagrams describe objects in a process. This was not exactly the case, as there was no real process in the DW presentation to describe. The DW bus matrix is a nice technique to get a top down view of the proposed DW. The concept is very simple to understand."	N/A

Table 8-4 (Continued) Analysis phase on development of the DW using OO techniques

From the researcher's point of view, the dimensional model, fact table and hierarchy analysis documents provided enough information to implement the physical tables on the database (DB), although no OO concepts were used. The technical architecture was purely DW based and did not incorporate any OO concepts.

Question	Business Analyst	Application support manager
	Design	
What are your comments on the dimensional model techniques used?	"The dimensional model provides a clear presentation of a business process." The concept of having a table (dimension) for each entity that describes a fact helps to "answer business questions, as well as to ask new questions about information in a certain business process that business never asked before."	N/A

Table 8-5 Design phase on development of the DW using OO techniques

How did you find the techniques used to design the ETL in the data staging services?	The "extract is defined very well with the help of the source to target mapping, ER and the business rules." The approach to use a high level to detail level diagram extract helped to provide an understanding of what processes need to go into the state chart diagram. "The application of the state chart diagram to an ETL process came very naturally and was not confusing at all." "The same can be said about the collaboration model that was created for the DW."	N/A

Table 8-6 (Continued) Design phase on development of the DW using OO techniques

From the researcher's point of view, the dimensional model techniques provided enough information for development. As with all OO development, a form of iteration was used between the analysis and the design phase. An iterative development lifecycle approach is a typical characteristic of OO development. The design techniques used in the ETL, such as the state chart diagrams, entity relational (ER) diagrams and collaboration diagrams are typical techniques used in normal OO development. The above mentioned techniques were well suited to effectively designing the DW.

The implementation of the DW did not involve the business analyst or the application support manager. Therefore, the discussion on the implementation is based on the researcher's experience. The implementation of the DW presented no problems. The design documents provided sufficient information for implementation. Unit testing was possible, as the DW implementation was grouped in packages. These packages were defined from the DW matrix and the state chart diagram.

Question	Business Analyst	Application support manager
	Testing	
What are your comments on the testing of the DW?	"Testing was possible on the outputs of the DW. Outputs in this case are the reports and ah hoc SQL query results. Each report was tested by comparing it to a similar report running production. Only *ad hoc* queries that are frequently used in the current reporting environment were testable. Unfortunately, *ad hoc* queries that did not have a similar report were not comparable. These queries were tested using different approaches where applicable." "Currently, most of the production systems have test packs to test the functionality of the application. Because the DW is a new system, test packs still need to be created. These test packs will automate the testing method mentioned earlier." "OO has different testing strategies for code testing. The only type of testing that was evident was unit testing. This did work well."	N/A

Table 8-7 Testing phase on development of the DW using OO techniques

8.2.5. Conclusion on the evaluation of the data warehouse

Based on user acceptance testing, incident requests and change requests generated by application support, there is evidence of the DW being used, while change requests also ensuring its growth. The following is a summary of the evaluation of the development approach focusing on the subjects below:

- Requirements gathering
- Analysis of the requirements
- Design

- Implementation
- Testing

8.2.5.1. Requirements gathering

Essential use case diagram

The perception of the essential use case diagram was favourable, as it provided a high-level overview of the business components available. The parties involved in requirements gathering were familiar with essential use case diagrams, therefore very little was needed to explain the concept of DW and business process (BP) use case diagrams.

Essential use cases

The DW and BP use cases were easily understood from the respective diagrams. Documentation on inputs and outputs, as well as possible grain needed by the DW use case, revealed no issues. There was also no difference between the BP use case and the normal OO development use case.

CRC diagram

CRC diagramming was not perceived favourably as use case diagrams. This technique was successful in identifying objects in the domain and the relationship requirements between these objects, but the general feeling was that there are better techniques available for identifying the domain objects and their relationships.

Gathering supplementary requirements

The documents gathered as supplementary requirements were no different from that of normal OO development.

Conclusion on requirements gathering

The use case diagram and use cases with the concept of splitting them into DW and BP use cases worked very well. This is largely due to the fact that the participants had background knowledge of OO development.

Some negative feedback on the use of CRC as a tool to define the domain model was received. This however, can be changed by using another domain model technique. The supplementary documentation used was standard to OO development. In general, the OO requirements gathering techniques worked very well for acquiring the necessary DW analysis information.

8.2.5.2. Analysis

System use cases

Like essential use cases, the systems use cases also followed the structure of DW and BP systems use cases. This also worked very well with the only negative feedback that the process of creating an essential use case and then a systems use case is time consuming.

Sequence diagramming

The sequence diagrams were not perceived as very favourable. The general feeling was that this technique was not suited to this type of analysis. The aim was to discover relationships and interaction between objects.

The data warehouse bus architecture

The use of data warehouse bus architecture matrix followed easily with the help of the objects identified (added as dimensions) and the DW use cases (added as subject areas). To a certain degree, the sequence diagram identified the interaction between the dimensions and the subject areas in the DW bus architecture matrix. It can therefore be concluded that the identified domain objects, the use cases and sequence diagrams (all of which are OO techniques) complement the development of the DW bus architecture matrix.

The dimensional table analysis

The dimensional table analysis is a DW technique and uses the DW bus architecture matrix as a starting point. OO concepts such as inheritance or polymorphism cannot be applied to the dimensional table analysis, as the industry standard for DB is relational and not OO based.

The fact table analysis

The fact table analysis is also a DW technique using the DW bus architecture matrix as a starting point. No OO concepts can be applied, for the same reason as for dimensional table analysis.

The hierarchy analysis

The dimensional table analysis is a DW technique, but hierarchies are identified during the BP system use cases. It can therefore be stated that the hierarchies are identified by the BP system use cases and the dimensional table analysis.

Technical architecture analysis

The technical architecture analysis consists of DW techniques only, none of which is OO based. This could be due to the type of DB the DW runs on, which is a relational DB and not OO based.

Conclusion on the analysis of requirements

Systems use cases and sequence diagrams were used to create the DW architecture bus matrix. Hierarchies were also created by using the systems use case and dimensional table. This suggests that the flow of analysis starts with OO techniques and ends with DW techniques. Owing to the nature of relational DB, it is not possible to implement OO concepts in the DW.

8.2.5.3. Design

Table designs

The design of the DW's physical tables is not OO based, since the type of DB used. It is therefore imperative to comply with relational DB table designs. No comment can be made about the DB size and index plan, as these are used for operational purposes and not for development. The OO concept of iteration was done between analysis and design and even the requirements of the DW.

Data staging designs

Based on the information of the analysis documents (dimensional and fact table designs, ER model and schematic plans), it was possible to design the data staging with OO techniques, such as state chart models and collaboration models. Also ER models were used to achieve this. The combination of these techniques was favoured and proved successful.

Conclusion on the designs

The use of OO concepts in the design of a DW running on a relational DB is not always possible. However, in data staging, OO techniques can be used to describe the process of the ETL package, although relational DB techniques are still required to describe the storage format.

8.2.5.4. Implementation

Implementation of the code was possible with the design documents provided. The code was implemented in a packaged fashion. This allowed unit testing and some form of reuse. Reuse forms, such as polymorphism, extends and inheritance were not possible due to the type of DB used.

8.2.5.5. Testing

Unit testing was possible, as each unit test consisted of duplication testing, reconciliation testing (where applicable) and changing dimension testing (where applicable).

Based on the evaluation and feedback of parties involved, the implementation of the DW may be considered a success.

8.3. Specifying learning

This section deals with specifying learning during evaluation of the DW. This forms part of the specifying learning phase of action research as explained in Baskerville (1999:13) and illustrated in Figure 8-3.

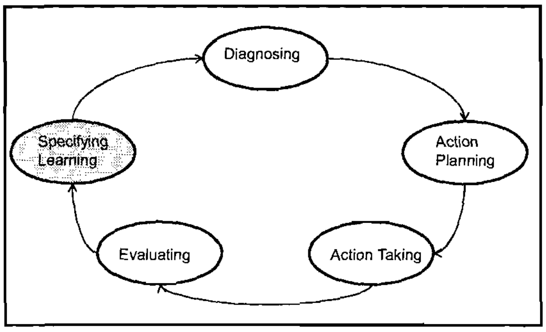


Figure 8-3 Specifying learning in the Action Research Cycle (Baskerville, 1999:14)

The following is a discussion on specifying learning, focusing on the following:

- Requirements gathering
- Analysis of the requirements
- Design
- Implementation
- Testing

8.3.1. Requirements gathering

The introduction of a DW use case diagram and DW use cases appeared to be successful. A comment was made about the necessity of DW systems use cases, the perception being that the information contained in the DW systems use case is the same as in the essential DW systems use case, only with data type definitions still to be added to the essential systems use case. For future iterations, one may consider executing the DW systems use case in the way commented on, but bearing in mind that both the essential use case and the systems use case are systems dependent.

Another comment on the DW essential use case was that it did not describe a sequence of events, but rather the data existing in the use case. The concept behind the DW essential use case is to discover which information is available and at what level. Although it worked very well, for future iterations consideration should be given to using another technique portraying what information is available within the scope of the business unit.

The concept of a business process use case diagram appeared to be working well, as this is essentially a normal OO use case diagram with the function of discovering the business processes within the department. For future iterations, this should be kept unchanged.

Some negative feedback was received regarding the use of CRC as a tool for defining the domain model. However, it should also be borne in mind that the insurance company does not use CRC as a domain modelling tool and that this could be the reason for rejection. For future iterations, consideration should be given to using another domain modelling technique, or to improve on educating the parties involved.

The supplementary documentation that was used is standard to OO development. In general, OO requirements gathering techniques worked very well for acquiring necessary DW analysis information and should be used for future iterations of the study.

The general perception was that the requirements gathering techniques were successful, even with negative feedback on the CRC technique.

8.3.2. Analysis of the requirements

As discussed above, the DW systems use case was perceived as tedious.

Negative feedback was received on the sequence diagrams. The general perception was that it is not suitable for this type of analysis. The interview suggested that the idea of a sequence diagram is to describe the flow of a process and the interaction between the defined objects on a systems level. A comment was made that the sequence diagrams created in this study were on a higher level. For future iterations, another process technique linking the objects and the process steps may be considered.

The DW bus architecture matrix was perceived to be successful. The fact that it is done iteratively, suggested an OO approach to the development of the DW bus architecture matrix. It was also stated that the DW bus architecture matrix was complemented by the OO techniques used for requirements gathering and analysis. These techniques should be used for future iterations.

The dimensional table, fact table, hierarchy analysis and technical architecture were perceived to be successful, but did not incorporate any OO techniques. The main reason for this being the underlying relational based DB system, preventing the application of OO concepts to the techniques used. If available, an OO DB should be considered for the next iteration of the study.

In general, the OO techniques did complement the analysis of the DW. A transition from OO structure (the use cases and sequence diagrams) to a more relational structure (dimensional tables, fact tables and hierarchy analysis) was perceived.

8.3.3. Design

As with dimensional modelling, design was purely relational-based, mainly because of the underlying platform (DB) being relational. Therefore, incorporation of any OO techniques could not be expected. Owing to its non-availability, an OO DB could not be used for this study, but may be considered for future iterations.

The design of the ETL was complemented by the state chart diagrams, ER diagrams and collaboration diagrams as it represents a process of transformation of data between two systems. These techniques should be used for future iterations of the study.

8.3.4. Implementation

Feedback on the implementation of the DW is based on the researcher's experience. The implementation was possible with the requirements and analysis documents received. A form of iteration was present between implementation and analysis and sometimes the requirements documents. This is characteristic of OO development.

The implementation process cannot be changed, but it can be made more efficient by means of optimising the analysis documents. Therefore, the implementation phase will be the same for future iterations.

8.3.5. Testing

Testing was completed by the business analyst, the interview suggesting that unit testing was possible and that automated testing can be applied to outputs of the DW. The only shortcoming during the testing phase was that the results of new *ad hoc* queries were not verifiable. Unit testing is a form of an OO technique, but more testing techniques should be considered for future iterations.

8.3.6. Future iterations of the study

Future iterations of the study are possible, but for financial reasons not feasible. Information systems, especially DW systems, are expensive to develop, however this study can be used for future research.

8.4. Conclusion

The research question considered in this study was to determine whether a data warehouse can be developed by using a data warehouse methodology and incorporating object-oriented techniques.

The development of the DW generally appeared to be successful.

The following are findings of the research question, as well as an evaluation thereof, as discussed in this chapter.

The use of OO requirements gathering techniques generally proves to be successful. The use case diagram and use cases with the concept of splitting them into DW and business process use cases worked very well. This is largely due to the fact that the participants had background knowledge of OO development.

An analysis flow starting with OO techniques and ending with DW techniques was suggested. The systems use cases and sequence diagrams were used to create the DW architecture bus matrix. Hierarchies were also created by using the systems use case and dimensional table.

The use of OO concepts in the design of a DW running on a relational database is not always possible. However, in the data staging process OO techniques can be used to describe the process of the ETL package, although relational DB techniques are still required to describe the storage format.

The only concepts of OO in the implementation process were the modular implementation of the code developed and the reuse (exact reuse). Other OO concepts, such as extends, inherits and polymorphism could not be introduced as the DW was developed on a relational DB.

The only concept of OO testing possible, was unit testing, but automated testing can be done.

The OO concept of iteration was done between analysis and design and even the requirements of the DW.

The use of a relational DB restricted OO in the design and implementation phases of DW development. Researching the use of OO DB management systems in DW development may therefore be considered.

One limitation was that the results obtained from this study were based on the experience of the parties involved. With no OO background available, the result could have been different.

8.5. Summary

The aim of the study was to explore the use of object-oriented methodologies in data warehouse development. The study followed a qualitative research approach and action research as the research method. Chapters 3, 4 and 5 discussed the literature needed for this study.

Chapter 3 explained what information systems development methodologies are. These are defined as a combination of systems development approaches (ISDA), systems development process models, systems development methods (ISDM) and systems development techniques.

A discussion on general OO methodologies followed in chapter 4. The following methodologies were discussed in detail:

- Object-Oriented Analysis (OOA)
- Object-Oriented Software Process (OOSP)
- Rational Unified Process (RUP)
- Object Modeling Technique (OMT)

A table was derived from common techniques and phases in methodologies listed above.

Chapter 5 was a literature study on the development methodologies for data warehouses. This chapter covered the business development lifecycle approach to data warehouse development (Kimball *et al.*, 1998) and data driven methodologies (Inmon, 1996).

The remainder of the chapters was structured around the Action Research Cycle (Baskerville, 1999), illustrated in Figure 8-4.

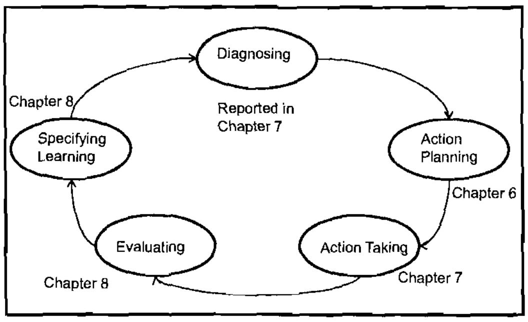


Figure 8-4 Chapters according to the Action Research Cycle (Baskerville, 1999:14)

Chapter 6 focused on the action planning phase of the Action Research Cycle. This chapter attempted to create a method for implementing a data warehouse by using methods and techniques available in both object-oriented methodologies and data warehouse methodologies.

Chapter 7 represented the action taking phase of the Action Research Cycle. This chapter was an interpretive experiment discussing the implementation of the method created in chapter 6.

The first part of chapter 8 depicted the evaluating phase of the Action Research Cycle. This part attempted to discover the perception of techniques used to implement the DW (chapter 7).

The second part of chapter 8 served as the specified learning phase of the Action Research Cycle. This part discussed the knowledge gained from the study, as well as possible recommendations for future iterations. Chapter 8 ended with the conclusion on the study.

References

AMBLER, W.P. 2001. The Object Primer 2nd ed.: New York: Cambridge University Press.

ANONYMOUS. 2007. Design of the data warehouse: Kimball Vs Inmon. [Web:] http://www.exforsys.com/tutorials/msas/data-warehouse-design-kimball-vs-inmon.html [Date of use 31 July 2007]

ANONYMOUS. 2006. Data Warehouse portal basics.
[Web:] http://www.dmreview.com/channels/dw_basics.html [Date of use 1 July 2006]

ANAHORY, S. & MURRAY, D. 1997. Data warehousing in the real world: a practical guide for building decision support systems. Harlow, England; Reading, Mass: Addison-Wesley.

AVISON, D.E. & FITZGERALD, G. 2003. Information Systems Development 3rd Edition: Mc-Graw Hill.

BASKERVILLE, R.L. 1999. Investigating information systems with action research. *Communications of the AIS*, vol 2 article 19.

BOAHENE, M. 1999. Information systems development methodologies: Are you being served? *Proceedings of 1999 Australasian Conference on Information Systems*. Wellington, New Zealand, p.88-99.

BOOCH, G. 1994. Object Oriented Design with Applications (2nd ed.): Benjamin-Cummings.

BOOCH, G., RUMBAUGH, J. & JACOBSON, I. 2001. The Unified Modelling Language Reference Manual: Addison Wesley Longman.

BRINKKEMPER, S., LYYTINEN, K. & WELKE, R.J. 1996. Methood engineering: Principals of Construction and Tools Support. London: Chapman & Hall,

BURRELL, G. & MORGAN, G. 1979. Sociological paradigms and organizational analysis. London: Heinemann.

CHUA, W.F. 1986. Radical Developments in Accounting thought. *The Accounting Review*, 61:615

COAD, P. & YOURDON, E. 1991. Object oriented analysis, 2nd Edition. New Jersey: Prentice Hall, England Cliffs.

HARVEY, L. 1990. Critical social research. London: Unwin Hyman.

HUISMAN, M. & IIVARI, J. 2003. Systems development methodology use in South Africa. *Ninth Americas Conference on Information Systems*, 1040-1052

IIVARI, J., HIRSCHHEIM, R. & KLEIN H.K.1998. A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. *Information Systems Research*, vol. 9. no 2

IIVARI, J., HIRSCHHEIM, R. & KLEIN, H.K. 1999. Beyond Methodologies: Keeping up with Information Systems Development Approaches through Dynamic Classification. *Proceedings of the 32nd Hawaii Conference on Systems Sciences*

IIVARI, J., HIRSCHHEIM, R. & KLEIN, H.K. 2000. A Dynamic Framework for Classifying Information Systems Development Methodologies and Approaches. *Journal of Management Information Systems*, 17(3):179-218

INMON, W.H. 1996. Building the data warehouse 2nd edition: John Wiley and sons.

JACOBSON, I., BOOCH, G. & RUMBAUGH, J. 2001.The Unified Software Development Process. Boston: Addison Wesley.

KAPLAN, B. & MAXWELL, J.A. 1994. Qualitative Research Methods for Evaluating Computer Information Systems. (*In*, ANDERSON, J.G. AYDIN, C.E. & JAY, S.J. (*eds*.). 1994. Evaluating Health Care Information Systems: Methods and Applications. CA: Sage, Thousand Oaks. p. 45-68.)

KIMBALL, R. & CASERTA, J. 2004. The Data Warehouse ETL Toolkit. Indianapolis: Willey.

KIMBALL, R., REEVES, L., ROSS, M. & THORNTHWAITE, W. 1998. The data warehouse lifecycle toolkit. New York: Wiley.

KLEIN, H.K. & MYERS D.M. 1999. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 23(1):67-94.

KUHN, T.S. 1970. The structure of Scientific Revolutions, 2nd edition. Chicago: University of Chicago Press.

LEEDY, P.D. & ORMOND, J.E. 2005. Practical research: planning and design: Pearson/Merrill Prentice Hall

MARCH, S. & SMITH, G. 1995. Design and Natural Science Research on Information Technology. *Decision support systems* 15(4):251-266.

MARTIN, P.Y. & TURNER, B.A. 1986. Grounded Theory and Organizational Research. *The Journal of Applied Behavioral Science*, 22(2):141-157.

MOORE, J.M. & BAILIN, S.C. 1988. Position paper on domain analysis. Lauerel, MD: CTA

MYERS, M.D. 1997. Qualitative Research in Information Systems. *MIS Quarterly*, 21(2):241-242. [Web:] http://www.misq.org/discovery/MISQD_isworld/. [Date of use 8 December 2008]

ORLIKOWSKI, W.J. & BAROUDI, J.J. 1991. Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research* 2:1-28.

RAMAKRISHNAN, R. & GEHRKE, J. 2003. Database Management Systems (third edition): McGraw Hill.

RAPOPORT, R.N. 1970. Three Dilemmas in Action Research. *Human Relations*, 23(6): 499-513.

ROB, P. & CORONEL, C. 2002. Database Systems Design, Implementation, and Design (5th Edition): Course Technology

RUMBAUGH, J., BLAHA, J., PREMERLANI, W., EDDY, F. & LORENSEN, W. 1991. Object-Oriented Modelling and Design. Englewood Cliffs, NJ: Prentice Hall.

SEN, A. & SINHA, A.P. 2005. A Comparison of using Data Warehousing Methodologies. *Communications of the ACM*, 48(3):79-84.

WALSHAM, G. 1993. Interpreting Information Systems in Organizations. London: Wiley.

WYNEKOOP, J.L. & RUSSO, N.L. 1993. Systems Development Methodologies: Unanswered questions and the research-practice gap. (*In* DEGROSS, J.I. BOSTROM, R. ROBEY, D. (*ed.*) 1993. *Proceedings of the Fourteenth International Conference on Information Systems*, Orlando, FL. p.181-190.)