THE EXPLORATION OF A CATEGORY THEORY-BASED VIRTUAL GEOMETRICAL PRODUCT SPECIFICATION SYSTEM FOR DESIGN AND MANUFACTURING

Yuanping Xu

A thesis submitted to the University of Huddersfield
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

School of Computing and Engineering
University of Huddersfield

February 2009

ACKNOWLEDGEMENT

First of all, I would like to express my sincere gratitude to my director of studies, Prof. Xiangqian Jiang for setting up this new research direction and the fundamental research frame. Many thanks also give to her for kindly help, advice and encouragement.

Great appreciation also goes to my second supervisor, Dr. Zhijie Xu, for his exceptional support and guidance throughout the project. He gave lots of time helping me to improve my academic writing and the capability of carrying out professional researches.

Many thanks give to Prof. Paul Scott in Taylor Hobson Ltd. for his valuable guidance and in-depth knowledge in Category Theory and verifying the mathematics.

I would like to thank Dr. Yan Wang, in providing data and case studies to test the VirtualGPS system. I also would like to thank Dr. Jonathan Pickering for proof reading this thesis.

I would like to show my appreciation to the Centre for Precision in University of Huddersfield, the School of Computing and Engineering and the Computer Graphics and Image Processing (CGIP) Research Group in Informatics department for providing me the opportunity and facilities to carry out this project.

A great deal of consideration and thanks must go to my parents, without my mother and father's belief, encouragement, education and financial support, this thesis can never be finished.

Finally, I would like to thank all my friends in Centre for Precision Technologies (CPT), CGIP and undergraduate classmates in Chengdu city of China for their kindly encouragements.

ABSTRACT

In order to ensure quality of products and to facilitate global outsourcing, almost all the so-called “world-class” manufacturing companies nowadays are applying various tools and methods to maintain the consistency of a product’s characteristics throughout its manufacturing life cycle. Among these, for ensuring the consistency of the geometric characteristics, a tolerancing language − the Geometrical Product Specification (GPS) has been widely adopted to precisely transform the functional requirements from customers into manufactured workpieces expressed as tolerance notes in technical drawings. Although commonly acknowledged by industrial users as one of the most successful efforts in integrating existing manufacturing life-cycle standards, current GPS implementations and software packages suffer from several drawbacks in their practical use, possibly the most significant, the difficulties in inferring the data for the “best” solutions. The problem stemmed from the foundation of data structures and knowledge-based system design. This indicates that there need to be a “new” software system to facilitate GPS applications.

The presented thesis introduced an innovative knowledge-based system − the VirtualGPS − that provides an integrated GPS knowledge platform based on a stable and efficient database structure with knowledge generation and accessing facilities. The system focuses on solving the intrinsic product design and production problems by acting as a virtual domain expert through translating GPS standards and rules into the forms of computerized expert advices and warnings. Furthermore, this system can be used as a training tool for young and new engineers to understand the huge amount of GPS standards in a relative “quicker” manner.

The thesis started with a detailed discussion of the proposed categorical modelling mechanism, which has been devised based on the Category Theory. It provided a unified mechanism for knowledge acquisition and representation, knowledge-based system design, and database schema modelling. As a core part for assessing this knowledge-based system, the implementation of the categorical Database Management System (DBMS) is also presented in this thesis. The focus then moved on to demonstrate the design and implementation of the proposed VirtualGPS system. The tests and evaluations of this system were illustrated in Chapter 6. Finally, the thesis summarized the contributions to knowledge in Chapter 7.

After thoroughly reviewing the project, the conclusions reached construe that the entire VirtualGPS system was designed and implemented to conform to Category Theory and object-oriented programming rules. The initial tests and performance analyses show that the system facilitates the geometric product manufacturing operations and benefits the manufacturers and engineers alike from function designs, to a manufacturing and verification.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

ACRONYM LIST

ACID	Atomicity, Consistency, Isolation, Durability
ADO	ActiveX Data Objects
ADT	Abstract Data Type
AFM	Atomic Force Microscope
ANSI	American National Standards Institute
API	Application programming interface
BCNF	Boyce-Codd Normal Form
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CASE	Computer Aided Software Engineering
CAT	Computer Aid Tolerance
CEN	European Committee for Standardization
CFI	CAD Framework Initiative
CGIP	Computer Graphics and Image Processing
CLI	Call Level Interfaces
CMM	Coordinate Measurement Machine
CORBA	Common Object Request Broker Architecture
CPT	Centre for Precision Technologies
DBMS	Database Management System
DB4O	Database For Objects
DDL	Data Definition Language
DML	Data Manipulation Language
DOM	Document Object Model
DTD	Document Type Definition
ER	Entity-Relationship
ERS	Empirical Relational System
FDE	Factorial Designed Experiment
GD&T	Geometrical Dimensioning and Tolerancing
GPL	General Public License
GPS	Geometrical Product Specification
HTML	HyperText Markup Language
ICT	Information and Computer Technology
IDE	Integrated Development Environments
ISO	International Organization for Standardization
JavaCC	Java Compiler Compiler
JDBC	Java Database Connectivity
JDO	Java Data Objects
MIT	McCarthy developed LISP at Massachusetts Institute of Technology
NF	Normal Form
NQ	Native Queries
NRS	Numerical Relational System
ODBC	Open Database Connectivity
ODL	Object Definition Language
ODMG	Object Data Management Group

ODP	Open Distributed Processing
OID	Object Identity
OIF	Object Interchange Format
OMG	Object Management Group
OQL	Object Query Language
PCTE	Portable Common Tool Environment
PDA	Personal Digital Assistant
PDF	Portable Document Format
P/FDM	Prolog/Functional DATA Model
PRIMA	Manufacturing Process Information Map
SDK	Software Developer Kit
SDN	Sun Developer Network
SEM	Scanning Electron Microscope
SME	Small Medium–sized Enterprises
SODA	Simple Object Database Access
SQL	Structured Query Language
STEP	Standards for Exchange of Product
STM	Scanning Tunnelling Microscope
SWT	Standard Widget Toolkit
TC	Technical Committee
TDC	Top-Dead Centre
TR	Technical Report
UI	User Interface
UML	Unified Modelling Language
USDP	Unified Software Development Process
W3C	The World Wide Web Consortium
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation

CHAPTER 1 INTRODUCTION

1.1 Project Introduction

In modern industries, manufacturers are applying various tools and methods to ensure the consistency of geometric characteristics for the machining products through the manufacturing life-cycle. To ensure the consistency of geometric characteristics and to facilitate global outsourcing, a universally accepted tolerancing language should be adopted to precisely transform functional requirements into manufactured workpieces and parts based on: mathematical rules and methods, consideration of macro and micro geometry, possibilities for measuring of quantities (especially tolerance quantities) and evolution of uncertainty, etc (Durakbasa and Osanna, 2001 [1]). The Geometrical Product Specification (GPS) is the modern and updated symbolic language that is used for specifying the functional requirements in technical drawing (Bennich and Nielsen, 2005 [2]). It is a standardized tolerancing language, which contains a set of standards organized in matrices. Therefore, some researchers refer to GPS as the GPS matrix system. It has been reported that GPS can save up to 15% in manufacturing cost through reducing misunderstandings and the ambiguity in defining the tolerance requirements (Humienny et al., 2001 [3]).

The initial GPS standards were set up by the International Organization for Standardization (ISO) to determine geometrical features of workpieces, such as size, distance and radius (Durakbasa and Osanna, 2001 [1]). It can also be used to verify workpieces according to their specifications as well as to suggest the measuring instruments and their calibration methods (Humienny et al., 2001 [3]). A number of important factors considered in this process include macro and micro geometry, quantity measures, uncertainty, measurement traceability and so on. In order to further optimize manufacturing resources through the scientific and economic management of various production processes and satisfy all the customized requirements of a product, the next generation GPS standards aim to integrate all the essential steps and data of a production practice in terms of their properties, such as the top-down or bottom-up manufacturing processes in the macro or nano scale production have been developed (ISO/TR 14638, 1995 [4]; ISO TC/213, 2001 [5]; Wang et al., 2004 [6]). However, the current GPS standards are over complex, abstract, and theoretical for many Small Medium–sized Enterprises (SME) in the manufacturing industry for following reasons:

- SMEs often lack GPS expertise. The inner relationships between different GPS matrices are vaguely defined. Hence, only the GPS experts are capable of cross-referencing and interpreting them to satisfy specific user requirements. For many SMEs subcontracting for large companies, these difficulties pose a great financial burden.
- It is difficult for users to apply tolerances on drawings to unambiguously express the functional requirements, or to interpret the symbolic language of GPS into various mechanical requirements. The failure to teach these skills leads to the vast majority of drawings used in industry today are ambiguous and can not communicate the true functional requirements of parts (Bennich and Nielsen, 2005 [2]). Therefore, the incorrect and ambiguous definitions of GPS requirements bring high economical risks to industry.
- GPS standards are often stored in text-based electronic file formats (e.g. PDF) organized by matrices, which are difficult for users to search and access without knowing specific search criteria. It is even more difficult for the application of computer-based knowledge inference processes.
- There are no existing de facto knowledge-based systems to manage this large maze of GPS standards and to maintain its data integrity and version consistency for GPS applications. Current efforts and pilot systems used to resolve these problems do not seem to provide mechanisms for GPS users to share data remotely; never mention to customise or even add their own new knowledge relating to certain processes.

As stated above, it is difficult right now to take the full advantages of these powerful and promising GPS standards to ensure the integrity of a specified product regarding its functionality, safety, dependability and interchangeability without fundamental renovations of their obsolete storage and access mechanism using the latest Information and Computer Technology (ICT).

It is envisaged that a knowledge-based information system for automatically implementing the GPS standards will facilitate the wider adoption of this tool in industry (Partridge and Hussain, 1995 [7]). During the last five years, various software systems have been developed to transform function-dependent demands into specifications of workpieces based on mathematical rules and methods. Unfortunately, almost all of them were based on the older technical standards with limited functional contents and a few simple technical specifications, which kept the system from automatically finding the appropriate GPS standards (Jiang, 2004 [8]). This drawback had hampered efforts to keep product specifications consistent with the GPS standards when the product function changes. Furthermore, the relational databases applied by all the current engineering aided design and manufacturing software tools can not support complex data structures to reflect the complicated relationships among parts and GPS standards, which are essential for comprehensive analysis and data manipulation to solve practical production problems.

To overcome the aforementioned usability problems, this project aims to develop a knowledge-based system framework called VirtualGPS system which focuses on developing an integrated GPS knowledge platform with knowledge generation and accessing facility based on the GPS matrices defined in GPS standards. Here, the term "virtual" refers to the effort in integrating the GPS information (especially these specified in the CEN and ISO standard documentation (ISO TC/213, 2001 [5])) and the corresponding GPS realization methodologies into a single framework regardless of their physical storage locations. At this stage, the system takes the surface texture as an example to demonstrate its functional features. It covered knowledge domains of GPS in dimensional and geometrical tolerances for surface and related manufacturing processes/equipments, verification principles, as well as uncertainty and measurement traceability. In future, it will enrich GPS knowledge domains for form, size and position. This has led to the emergence of the classical problem of storing and managing large amounts of data in various complex structures that are difficult or impossible to be divided into strict formats of flat table relations applied in relational Database Management Systems (DBMS). To solve the problem, the proposed VirtualGPS system applied an object-oriented approach based on Category Theory.

1.2 Aims and Objectives

The project aims to investigate a software solution (VirtualGPS) to handle the large amount of data in various complex structures relating to matrices defined in the GPS. The VirtualGPS system will focus on solving the design and production stage problems by acting as a "virtual" domain expert through translating GPS standards and rules into the forms of computerized expert advices and warnings. This system can also be used as a training platform for teaching engineers how to utilise unambiguous tolerance specifications for expressing functional requirements, and how to apply GPS to guide the integrated manufacturing and measurement processes. As a core part of the VirtualGPS, the project researched and devised a Category Theory-based object-oriented DBMS named categorical DBMS to utilize its capability for handling complex multi-level objects and object relationships, which is of vital importance in managing large scale geometrical product designs, manufacturing and measurement data.

It is envisaged that the research will contribute to the domain knowledge by providing a case study for incorporating state-of-the-art research advancements and technologies in the information/knowledge-based systems and database fields. Also it will provide a computerized system which can generate expert knowledge to integrate product functions, specifications of micro- and nano-geometry, manufacturing processes and verification procedures. The research objectives of this project can be classified as follows:

(1) To provide a unified knowledge acquisition and knowledge representation mechanism to retrieve and organize knowledge from various GPS documents.

(2) To identify suitable data structures for storing GPS knowledge within the final software system. It covers data from the functional, specification, manufacture and verification aspects in GPS field. The complex relationships between both areal and profile specification and metrology, such as profile and areal standards, filtration, parameter algorithms, instrumentation, measurement procedures, instrument calibration, and uncertainty will be investigated.

(3) To build a consistent and integrated framework to encompass the data gained in the process explained in the above point. It is anticipated that an object-oriented DBMS needs to be built to utilize the Category Theory-enabled abilities of querying and preserving complex objects and their relationships (often in the forms of arrows in the schema diagram). Moreover, this approach should facilitate in solving the complex database problems in object-oriented DBMSs, such as typing, message passing, view, and query closure. The categorical DBMS of this project should have both the flexibility for storing and implementing the complex objects and also has solid mathematical foundations with formal semantics.

(4) To provide a unified knowledge base for supporting engineering decisions in choosing appropriate GPS parameters according to the required functional performances.

(5) To enable an automated querying mechanism for guiding designers with relevant GPS specifications. For example, it can be used to provide correct and unambiguous geometrical specification (the technical specification) relevant to the functional design intent.

(6) To equip a rating and ranking inference engine for locating and retrieving GPS-recommended manufacturing processes and equipments.

(7) To link similar functions to aid decisions on measurement procedures and equipment.

(8) To achieve the system functions, the software specification has also included the following features:

- Client/Server structure for the synergy between geographically dispersed designers, production engineers and metrologists to work closely.
- User-friendly system interfaces for accessing system functions such as cross-referencing, reporting and updating.

At present, all of the above eight objectives are using the surface texture part of GPS as demonstrating and testing examples. After achieving the above eight objectives, the VirtualGPS system will enable non-experts to use GPS standards in an efficient manner. It will also ensure that when a product design changes, the relevant GPS specifications will be updated automatically to remain consistent with relevant GPS standards. Moreover, with the trend of globalisation in manufacture industries, the remote data access features and web-based user interfaces of the system will become the norm.

1.3 Project Approach

The project started with an extensive literature review of the state-of-the-art in GPS advancements, knowledge-based system evaluations, relational, object-relational and object-oriented database applications and data-mining practices. The project development approach has been demonstrated as follows:

(1) **Initial Design.** Based on the literature review of the problem domains and the analysis of user requirements on the VirtualGPS system, the overall system framework has been designed using the Category Theory. To address problems highlighted in Section 1.1, this project also decided to use the Category Theory to model knowledge structures and knowledge operations.

(2) **Proof-of-Concept/Prototype Development.** A proof-of-concept prototype

was developed in the Java language. This has facilitated the refinement and completion of the system architecture with improved understanding on some implementation issues. This system also served as a demonstration of the design concepts and capabilities of the final system with feedbacks from various tests.

(3) **Design Modification.** After collecting and analysing feedback from system tests, the system design on both the conceptual model and the prototype system was refined.

(4) **Development and Implementation.** The final system will be produced using Java and XML as development tools. Java was selected for its comprehensive functionality, sound stability and open-source nature, whilst XML technology is used to structure reports and communicate between manufacturing engineers. In this project, a native manipulation language was developed based on the Category Theory to match the so-called categorical object model adopted in this project.

(5) **Testing and Validation.** The proposed software system VirtualGPS is strictly tested and verified to evaluate its performance over existing solutions. In this project, tests were continuously being undertaken during every major phase to ensure that it has good functionality and stability. The diagram chasing and algebraic deductions based on Category Theory are used to ensure the integrity of knowledge base and database schemas of this system. Researchers and GPS experts in Centre of Precision Technology (CPT) in the University of Huddersfield were invited to test the software to assess whether the system can satisfy the demands from industry, as well as whether it meets the aims and objectives of the project. Further revisions for the proposed system might take place based on the feedback from these tests.

1.4 Thesis Structure

The following paragraphs provide a brief summary for the remaining chapters of this thesis.

Chapter 2 highlights the context of various problem domains relating to this project, which includes both engineering and computer science fields. For the engineering field, introductions of GPS and surface texture are given. For the computer science domain, detailed surveys over different database solutions, knowledge-based systems and XML technology, and their relationships to this project are discussed.

Chapter 3 focuses on developing a unified categorical modelling mechanism based on Category Theory for knowledge acquisition/representation, database schema construction and descriptions of the system frameworks. This chapter starts with a discussion on the necessary notions of Category Theory and rationales for using the Category Theory, and then provides a categorical object model for representations of knowledge and database schemas. A categorical software design process and an inference identifying square are also defined in this chapter for explaining the design of the whole software architecture and the modelling of knowledge inferences respectively. Moreover, examples for applying the categorical object model, the categorical software design process and the inference identifying square are illustrated in corresponding sections of this chapter.

Chapter 4 focuses on discussing the implementation of the categorical DBMS. This chapter starts with a discussion on why DB4O (database for objects) is chose as the implementing basis for the categorical DBMS. Then, it moves to explain the categorical architecture for the categorical DBMS, the necessary functional extensions for DB4O, and how to implement the categorical object model on the categorical DBMS. This chapter concludes with a demonstration of the visual management interface for the categorical DBMS.

Chapter 5 focuses on describing the design and implementation of the VirtualGPS system, which takes the surface texture as an example. The VirtualGPS contains four modules and each module in turn contains four components (sub-knowledge bases). The design of each module or component in the VirtualGPS system goes through a categorical software design process. After specifying the design of the VirtualGPS system, tools and platforms for implementing the system are discussed in this Chapter. This chapter concludes with a working case study to assess the design features and functionality of the system.

Chapter 6 focuses on discussing the tests and evaluations carried out on the categorical DBMS and the VirtualGPS system.

Chapter 7 deals with the final assessment of the project that focuses on the summary of its outcomes and contribution to knowledge. A discussion for the future work is also included at the end of this chapter.

CHAPTER 2 RESEARCH DOMAIN SURVEYS

This chapter focuses on introducing and explaining of the domain knowledge and its context that are heavily used in this project, which do not just originate in computer science but also come from the field of precision engineering. It covers: GPS and an overview of surface texture, evaluation of current Database Management Systems (DBMS), knowledge-based system review and eXtensible Markup Language (XML)/eXtensible Stylesheet Language Transformation (XSLT) descriptions. These research domains provide the foundations to develop the VirtualGPS system.

2.1 GPS and Surface Texture Overview

2.1.1 Introduction of GPS Framework

Traditionally, when a machining part is being designed, designers will only work on nominal specifications – that is workpieces expressed as ideal geometries without any geometrical errors, i.e., parts with perfect surfaces. However, any actual parts being produced in the real world will be far from perfect. Various deviations could exist in the forms of shape distortions, differences on dimensions and surface roughness, etc. Furthermore, the process of assembling parts is also error-prone where additional deviations easily occur, resulting in non-satisfying products. In a real production scenario, despite these deviations, a product may still be regarded as acceptable if the errors are properly controlled within certain limitations, which leads to the concept of tolerances. Therefore, there should have some standards to define these tolerances and ensure the real geometrical products are limited in certain extent of deviations. Moreover, during the last a couple of decades, manufacturing industry become more and more flexible and global through outsourcing. Geographically dispersed (remote) design and manufacturing practices are rapidly increasing (Humienny et al., 2001 [3]). This move is another major contributing factor to the generation of a set of universal standards and rules to unify the characteristics of workpieces using the so-called Geometrical Product Specification and Verification (GPS) Standards.

2.1.1.1 GPS Definition

The Geometrical Product Specification and Verification (GPS) matrix system is a tolerancing specification tool for expressing geometrical tolerances in technical drawings, which currently is the only worldwide symbol language available for communicating geometrical requirements (Bennich and Nielsen, 2005 [2]). The GPS is developed based on the Geometrical Dimensioning and Tolerancing (GD&T) with the addition of more detailed definitions of the requirements. This allows designers to express functional requirements much more precisely than before.

2.1.1.2 GPS Advantages

Users can gain technical, competitive, and economic advantages by using GPS as the tolerancing language in their drawings through following points (Bennich and Nielsen, 2005 [2]; Humienny et al., 2001 [3]):

- GPS drawings impose artificial constraints on manufacturing, which can help users to define non-functional parts. Therefore, GPS can save manufacturing cost through reducing work stoppages due to non-functional parts jamming assembly lines or lack of functional parts idling production.
- GPS can greatly improve communications between designers, manufacturers and metrologists. Therefore, savings come from reducing misunderstandings amongst the various roles involved in manufactures. This is very important for those companies that subcontract or outsource the manufacturing of parts to reduce unqualified products.
- GPS can quantify the ambiguity in a tolerance requirement when it is applied to a real part through specification uncertainty. This can be used to improve product designs.

2.1.1.3 The Framework of the GPS Standards

GPS aims to cover the whole spectrum of manufacturing design and production stages through specifying and verifying parts' sizes and dimensions, geometrical tolerances, and surface properties and to ensure the consistency of some essential properties of products no matter where they are designed and produced (Humienny et al., 2001 [3]). Generally, GPS standards are applied to ensure the following essential properties of products:

- Functionality. For example, if elements of a machine tool meet certain geometrical tolerances such as straightness of bebways, the machine can work properly.
- Safety. For example, the crankshaft pin is ground according to specifications concerning vibration to avoid fatigue cracking which will destroy the engine.
- Dependability. This is to guarantee the long work life of a machine.
- Interchangeability. This is to benefit new machine assembly and to facilitate

repair.

Figure 2.1 shows a simplified diagram of a typical GPS standards framework.

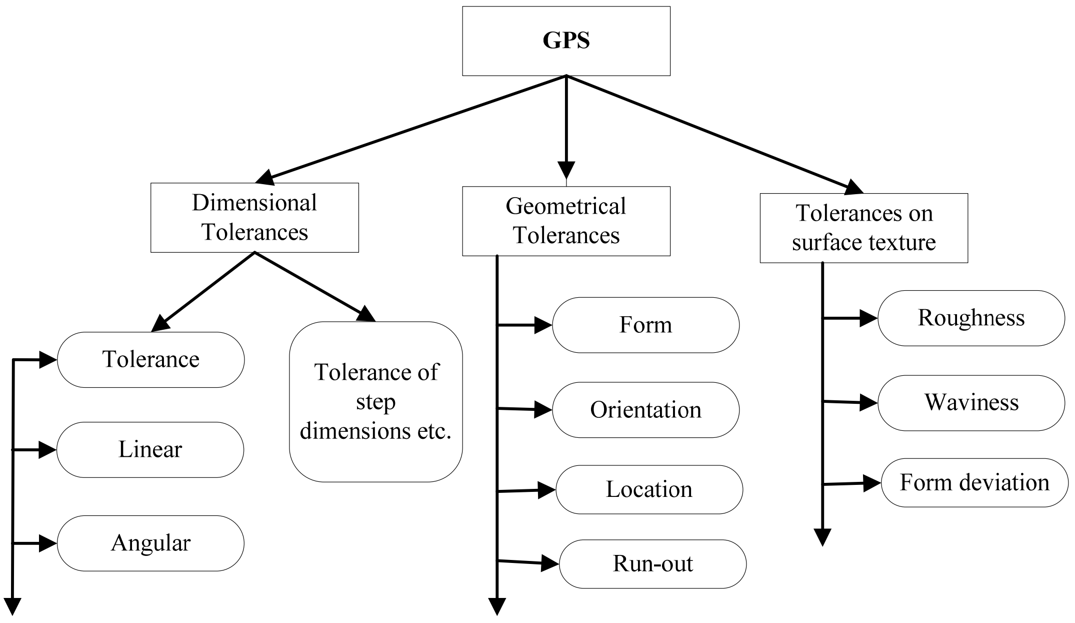


Figure 2.1: An Example of a GPS framework.

The key word "*tolerance*" in Figure 2.1 normally contains three parts: feature, characteristics and condition. Taking surface texture as an example, the feature is point, line or surface. It includes the integral feature and the derived feature. The integral feature is the surface or profile sections on a surface and the derived feature comprises the centrepoint, median line, median surface or offset feature from one or more integral features. A characteristic is the single property of one or more features expressed in linear or angular units. The features are described by characteristics, including different mathematical parameters and their numerical values, based on a set of data points from the features under consideration. Conditions are added to define acceptable limits for the measured value of a characteristic (tolerance values). Thus, these parts together can be used to determine the functional properties of a surface.

2.1.1.4 Forms of GPS Standards (Matrix)

According to the technical report ISO/TR 14638 published in 1995 (known as the Masterplan), the standards in GPS can be classified into fundamental GPS standards, global GPS standards, general GPS standards and complementary GPS standards (ISO/TR 14638, 1995 [4]). The general GPS standards are the kernel of the Masterplan. They are ordered in a matrix in which all the rows constitute 18 chains of standards in total (size, distance, radius, angle etc.) with each column defining various characteristics of geometrical features. Therefore, the whole GPS (ISO/CEN) is also called as the GPS matrix system (Humienny et al., 2001 [3]). Table 2.1 shows the chain of standards relating to size parameter that are grouped into six aspects: the product documentation indications, definition of tolerances, definitions of characteristics of actual feature, assessment of the workpiece deviations, measurement equipment and calibration requirements, and measurement standards. In an ideal case, based on each chain of standards, the process of manufacturing a part can be clearly defined by taking into factors such as setting up unambiguous specifications, and interpreting manufacturing specifications and verification information.

1	**2**	**3**	**4**	**5**	**6**
ISO 129 ISO 286-1	ISO 286-1 ISO 286-2	ISO 286-1 ISO 8015 ISO 14660-2	ISO 14253-1	ISO 463 ISO 9121 ISO 9493 ISO 10360-1 ISO 10360-2 ISO 13225 ISO 13385 ISO 14253-1	ISO 3650 ISO 14253-1

Table 2.1: The chain of standards relating to the "*size*" (the 1st row of the general GPS Matrix).

The latest version of general GPS matrix system is composed of 108 cells (6 by 18) and each of them contains at least one standard. In the future, there will be more standards to be filled into those cells. Hence the GPS matrix system will become more complex and difficult to be handled.

2.1.1.5 The Framework of Surface Texture

This project intends to research and develop a software solution that will provide a unified platform for designers and manufacturers to overcome these GPS application difficulties discussed in Chapter 1. It is intended to benefit industry by allowing the use of modern GPS standards, e.g. surface texture specification and verification.

As demonstrated in Section 2.1.1.3, the GPS covers three aspects: Dimensional tolerances, Geometrical tolerance and Tolerances on surface texture. Among them, surface texture is of the crucial state in the GPS. It represents the local deviations of a surface from its ideal shape in terms of roughness, waviness and form, which covers a wide spectrum of production activities, from the design function to specification on a drawing, from the manufacturing process to verification. It is an important factor in production for monitoring the production processes, preventing failures of the products, ensuring surface quality and inferring the functional performance of a surface. Due to its importance in ensuring the quality of the final product, surface characteristics are rigorously checked throughout the whole production lifecycle. Table 2.2 lists the GPS matrix chains relating to surface texture (Humienny et al., 2001 [3]).

Chain link number		1	2	3	4	5	6
Geometrical characteristic of feature							
14	Roughness profile	ISO1302	ISO 4287, 12085, 13565-1, 13565-2, 13565-3	ISO 4288, 12085, 11562, 13565-1	ISO 4288, 12085	ISO 3274	ISO 5436, 12179
15	Waviness profile	ISO1302	ISO 4287, 11562, 12085	ISO 11562, 12085	ISO 12085	ISO 3274	ISO 5436, 12179
16	Primary profile	ISO1302	ISO 4287, 11562, 13565-3	ISO 4288	ISO 4288	ISO 3274	ISO 5436, 12179

Table 2.2: Position of surface texture standards in the GPS matrix model.

As the prototype system taken surface texture as an example, the ISO chains in Table 2.2 provided foundation knowledge for the surface texture part of the VirtualGPS system (see Chapter 5).

2.1.2 GPS Application Difficulties

The aforementioned four major shortcomings relating to applying of ISO/CEN GPS standards indicate that there is need for development of "new" software systems to facilitate GPS applications. However, the current computer aided design and manufacturing software systems are still struggling to meet the demands of the global and dynamic manufacturing environments and fail to cope with the complexity of the whole GPS world due to the following reasons:

- Most systems do not provide precise drawing indication. For example, they have no function associate to drawing indication $\phi 30 \pm 0.1$.
- Different types of measurement methods may lead to very different results. The lack of effective communications has resulted in wide misunderstanding between the design concept and the real product. Experience has shown that the average costs resulting from such shortcomings of incomplete GPS technical documentation can amount to as much as 20% of the production turnover (ISO TC/213, 2002 [9]).
- Almost all current computer aided manufacturing software are based on the

old technical standards without applying the modern GPS, such as:

1) **STEP**: Standards for Exchange of Product Model Data (STEP) is a set of standardized protocols for computer-interpretable product information developed by ISO/TC 184/SC4. It relies on CAD vendors to provide translators that can read and write STEP formats. The product information defined by the STEP only defines how it can be exchanged between CAD/CAM systems using standardized protocols. It does not provide functions such as automatically generating product specifications based on GPS standards according to the required product functions. It also provides little support on issues like suggesting manufacturing processes, verification methods, and calibration equipments.
2) **Geometric Tolerance**: VisVSA (UGS, 2003 [10]) and CETOL (Sigmetrix, 2002 [11]) are popular commercial packages for tolerance calculation. The modern CAT (Computer Aid Tolerance) package (CATIA 3D Functional, VSA-GDT, VisQuality, and VisVSA) is implemented on CAD platforms (CATIA, UNIGRAPHICS, Pro/Engineer) that utilize solid modelling representation systems based on variation geometry. These CAT packages have functions such as associative intelligent, 3D tolerance specifications, annotation verification, and simulation/ prediction of manufacturing processes with variations at the assembly level.
3) **Limit and Fits**: There are a number of software implementations of the ISO standards for the so-called "Limits and Fits" being developed in various countries (examples can be found on www.hexagon.de). These software systems can decode the size limits specified by the tolerance classes into deviations to calculate the fit clearance/interference, and to determine the fit type as well as providing tolerance zone visualizations.
4) **CMM Software**: The latest CMM software permits interactive graphical inspection planning and programming based on the 3D CAD data with automation of the probe path generation (Jiang, 2004 [8]). CMM software such as Umess, Calypso (Zeiss), QUINDOS (Brown&Sharpe) and so on are often based on 3D CAD/CMM programming software for measuring, evaluating, simulating, curve-data importing, and model comparison (Humienny et al., 2001 [3]; Carl Zeiss Industrial, 2004, 2006 [12, 13]; Brown&Sharpe, 2006 [14]). For example, QUINDOS dimensional

measurement software provides flexibility for the inspection of prismatic parts such as engine blocks or gearboxes. Besides that QUINDOS can also measure and evaluate special shapes or geometry produced in today's industry (Brown & Sharpe, Inc., 2006 [14]). However, these metrology packages which set the measurement standard through direct and seamless integration with CAD data, rarely make assessment using ISO geometrical tolerance definitions (Humienny et al., 2001 [3]; Jiang, 2004 [8]). For example, when the perpendicularity of a cylinder axis to a plane is calculated, the associated derived axis is often used rather than the extracted derived axis required by ISO.

5) **Surface Texture Analysis System**. An internet-based surface texture analysis and information system, developed by Center for Precision Metrology in the University of North Carolina, claimed to solve the problem that current surface texture analysis systems are weak in developing process knowledge or mapping the observed effect to causes. For example, after taking several measurements on a workpiece, users can use traditional systems to filter the profile at a standard cut_off length and then get a table of calculated parameter values. However, these systems can not provide a documentation mechanism to store process parameters with metrology data for observing how process parameters relate to variability in the surface parameters. This system focuses on filter selection, filtering calculations and measured data analysis.

In general, the major software systems at present are still weak on functionality and relying on ambiguous dimensioning and tolerancing practices based on the nominal model methodology and geometry theory. Features such as product function, surface properties and the related verification principles, measuring equipment, calibration requirements, uncertainty and measurement traceability are often largely ignored. One of the major reasons for causing these drawbacks is that the traditional database systems applied by all the current engineering aided design and manufacturing software tools can not efficiently support complex data structures to reflect the complicated relationships among parts and GPS standards, which are essential for comprehensive analysis and data manipulations to solve practical production problems. The next section will discuss these classical traditional database system solutions with their advantages and disadvantages demonstrations.

2.2 Current Database System Solutions

2.2.1 Definitions and History

A database is a self-describing collection of integrated data with different structures designed to meet the information needs of an organization (Martin, 1977 [15]). A Database Management System (DBMS) is a compute program that controls the organization, storage and retrieval of data in a database and a database application is a compute program that interacts with a DBMS (Connolly and Begg, 2001 [16]). DBMS did not come into the market until 1960's and the first commercial DBMS − IMS appeared in early1970 (Lin, 2003 [17]). The modern DBMSs are raised to solve the problems of file systems, which support retrieving and storage of large amount of data in a computer (Molina, 2008 [18]). Researchers in the database field found that data has value and semantic meaning, so data models are required to be introduced to improve the reliability, security, efficiency of the access (Lin, 2003 [17]). Data models provide a way to describe what information is to be contained in a database, how the data organization of information is structured, and how the data will be related to each other for quick access and efficient management. As every DBMS has a data model behind it, so DBMSs can be classified into five basic types according to the data models that they are applying:

- Hierarchical DBMS (e.g. IMS)
- Network DBMS (e.g. CODASYL)
- Relational DBMS (e.g. MySQL, SQLServer)
- Object-relational DBMS (e.g. P/FDM)
- Object-oriented DBMS (e.g. ObjectStore, DB4O)

These data models describe not only the structure of the target databases but also the operations that can be performed on them. Each database has a "schema" that is a computing language description of its data model. Therefore, a data model often contains:

(1) Structure. The structure formed by classes, attributes, inter- or intra-associations. The structure is represented in both diagrammatic symbols and expressed as a schema by using a data definition language.

(2) Manipulation/Operation. Manipulation is formed by a query language in terms of searching, deleting or updating of the database.

(3) Rule. Rule is the restriction on the data model, for example, integrity

constraints.

Figure 2.2 shows the timeline of the history of these five major data models.

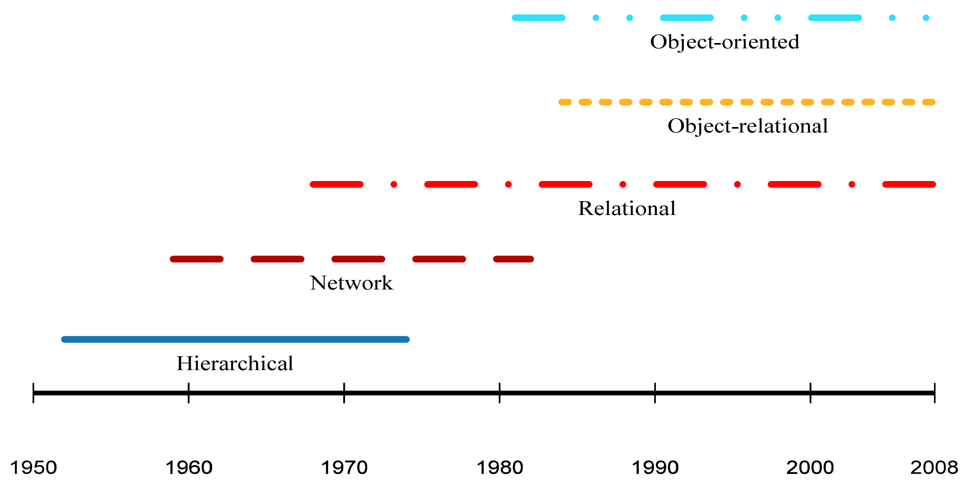


Figure 2.2: Timeline of the history of five data models (Tupil, 2008 [19])

The hierarchical and network databases had not become popular in modern database applications because of several fundamental limitations. For example, in hierarchical and network databases, data accesses are through low-level pointer operations to link records. Users need to know the physical database structure to query and update information. The first generation data models (hierarchical, and network) and the second generation (relational) are all record based and using simple data types, which have limited application supports. The third generation of data model (object-oriented) started in late 1980's, which can better support complex data types, having band to object-oriented programming languages such as Smalltalk, C++ and Java. All DBMSs contain its own Data Definition Language (DDL) and Data Manipulation Language (DML) (Cattell, et al., 2000 [20]). The DDL allows users to define their data types and interfaces. The DML allow programs to create, delete, retrieve and update the instances of those data types. In object-oriented DBMSs, the DDL is called Object Definition Language (ODL), which defines the characteristics of types (classes) including their properties and operations. For this thesis, the following sections concentrate on giving a detailed overview on relational, object-relational and object-oriented DBMSs.

2.2.2 Relational DBMS

In 1970, E.F. Codd proposed the relational model for databases that enabled database designers to focus on describing logic aspects (schema) of data without considering the physical storage strategies (Gray, 1992 [21]). Based on the relational model, a set of commercial products such as Oracle, DB2, MySQL and Sybase had been developed during the 1980's and 90's. Since then, the relational database has become the mainstream basis for high performance database applications.

2.2.2.1 Relational DBMS Standards

From the 1986, Structured Query Language (SQL) began to be widely used in relational DBMSs and in the same year, the American National Standards Institute (ANSI) standardized SQL (Devarakonda, 2001 [22]). This standard was updated in 1989, in 1992 (called SQL2), and again in 1999 (called SQL3). Standard SQL is sometimes called ANSI SQL or SQL92 and all major relational DBMSs support this standard but each has its own proprietary extensions (Stanezyk, 1993 [23]). Thus, the world wide accepted standard for relational DBMS formed by the SQL (containing both DDL and DML) and the relational data model. SQL includes statements for data definition, modification, querying and constraint specification.

2.2.2.2 Relational DBMS Overview

During the development of the last three decades, there are around 40s relational DBMS products developed by various vendors. This sub-section introduces three most popular DBMSs: Oracle, SQLServer and MySQL. The Table 2.3 gives a brief introduction on the background of these three relational DBMS.

Vendors	Latest Products	Started
Oracle Corporation	Oracle (8*i*)	1979
Sun Microsystems	MySQL(5.0.67)	1996
Microsoft	SQLServer (2005 SP2)	1989

Table 2.3: Current relational DBMSs.

The Table 2.4 shows a comparison of the basic characteristics of relational DBMSs.

Products	Oracle (8*i*)	MySQL(5.0.67)	SQLServer (2005 SP2)
Atomicity, Consistency, Isolation, Durability(ACID)	YES	YES	YES
Referential integrity	YES	YES	YES
Transactions	YES	YES	YES
Intersect	YES	NO	YES
Inner joins	YES	YES	YES
Outer joins	YES	YES	YES
BLOB	YES	YES	YES
Interface	SQL	SQL	SQL

Table 2.4: Comparison of the basic characteristics of three relational DBMSs (Wikipedia, 2006 [24]).

Table 2.5 shows the operating system supports of the three relational DBMSs:

Systems	Windows	Mac OS X	Linux	Unix
Oracle (8*i*)	YES	NO	YES	YES
MySQL(5.0.67)	YES	YES	YES	YES
SQLServer (2005 SP2)	YES	YES	No	No

Table 2.5: Comparison of operating system supports of three relational DBMSs.

The basic features in relational DBMSs are demonstrated as follows:

(1) The relational data model uses the Set Theory as its formal mathematic foundation.

(2) The relational model uses keys (primary key, component key) to uniquely represent data records whilst using use foreign keys to form relationships.

(3) Relational DBMSs use 2D "Tables" to represent entities and relationships between them.

(4) The "Tables" must obey the Normal Forms (such as, 3NF, BCNF).

(5) Relational DBMSs normally use SQL language to query and define data and data constraints.

2.2.2.3 Advantages

The advantages of relational DBMSs based on the relational data model can be summarized as:

(1) Relational DBMSs supported by the relational data model are based on matured and stable mathematic theory (i.e. Set Theory), which enables them to keep rigor integrity and have good reliability and changeability (Stanezyk, 1993 [23]). This is one of major reasons for the success of the relational DBMSs.

(2) The relational data model in relational DBMSs disconnects the conceptual data modelling with physical data storage and its access strategies.

(3) Relational DBMSs often support storage of large amount of data.

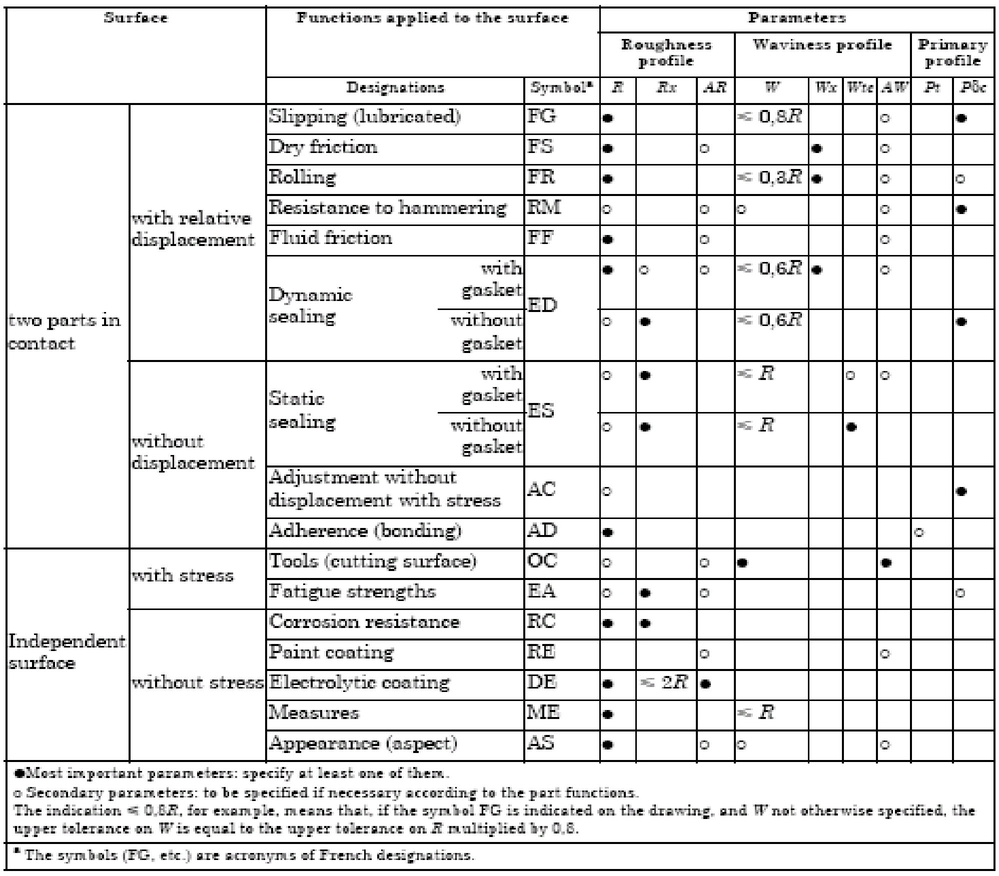
(4) Relational DBMSs have easy-to-use query, view, update, addition, deletion mechanisms. The SQL language and normalization rules can efficiently support these mechanisms.

(5) There are various powerful relational DBMS commercial products that can give database users other supplementary features such as storage plans, concurrency strategies, transaction managements, and backup strategies.

2.2.2.4 Disadvantages

The disadvantages of relational DBMSs are also prominent in the following areas:

(1) The relational DBMSs are limited in dealing with scientific applications and other applications that involve complex interrelationships of data. In order to minimize data redundancy, reduce design flaws and ensure the integrity of databases after addition, deletion, and modification on data sets, relational DBMSs must follow a certain Normal Forms (such as 3NF, BCNF), so they are weak in dealing with many-to-many relationships and other complex nested and embedded structures. It is unavoidable to face the tasks of storing and accessing those complex forms (as shown in Table 2.6):



Surface		Functions applied to the surface			Parameters								
					Roughness profile			Waviness profile				Primary profile	
		Designations		Symbol[a]	R	Rx	AR	W	Wx	Wte	AW	Pt	$P\delta c$
two parts in contact	with relative displacement	Slipping (lubricated)		FG	●			$\leqslant 0{,}8R$			○		●
		Dry friction		FS	●		○		●		○		
		Rolling		FR	●			$\leqslant 0{,}3R$	●		○		○
		Resistance to hammering		RM	○		○	○			○		●
		Fluid friction		FF	●		○				○		
		Dynamic sealing	with gasket	ED	●	○	○	$\leqslant 0{,}6R$	●		○		
			without gasket		○	●		$\leqslant 0{,}6R$					●
	without displacement	Static sealing	with gasket	ES	○	●		$\leqslant R$		○	○		
			without gasket		○	●		$\leqslant R$		●			
		Adjustment without displacement with stress		AC	○								●
		Adherence (bonding)		AD	●							○	
Independent surface	with stress	Tools (cutting surface)		OC	○		○	●			●		
		Fatigue strengths		EA	○	●	○						○
	without stress	Corrosion resistance		RC	●	●							
		Paint coating		RE			○				○		
		Electrolytic coating		DE	●	$\leqslant 2R$	●						
		Measures		ME	●			$\leqslant R$					
		Appearance (aspect)		AS	●		○	○			○		

●Most important parameters: specify at least one of them.
○ Secondary parameters: to be specified if necessary according to the part functions.
The indication $\leqslant 0{,}8R$, for example, means that, if the symbol FG is indicated on the drawing, and W not otherwise specified, the upper tolerance on W is equal to the upper tolerance on R multiplied by 0,8.

[a] The symbols (FG, etc.) are acronyms of French designations.

Table 2.6: Example of a classification of surface function together with a relationship table for motif parameters taken from ISO 12085 (ISO 12085, 1996 [25]).

For example, to store information from a complex matrix-style form as in Table 2.6, a relational DBMS has to divide the matrix into a number of smaller normalized tables linked via foreign keys, which may cause integration problems since a “join” operation has to be performed every time when queries are performed on the “Has-a” relationship between objects. Moreover, relational DBMSs also extremely inefficient at handling new data types such as images and video streams.

(2) Relational DBMS applications often suffer from "impedance mismatch" problem, which means that there is a large gap between object-oriented programming languages and relational DBMSs. For example, database applications use object-oriented programming languages to create and manage object instances that will have difficulties in converting them into table formats for storage and retrieving. The computing resources cost of relational DBMSs for converting very complex data structures will be dramatically increasing when the data volume becomes large. Figure 2.3 demonstrates the impedance mismatch in relational database applications:

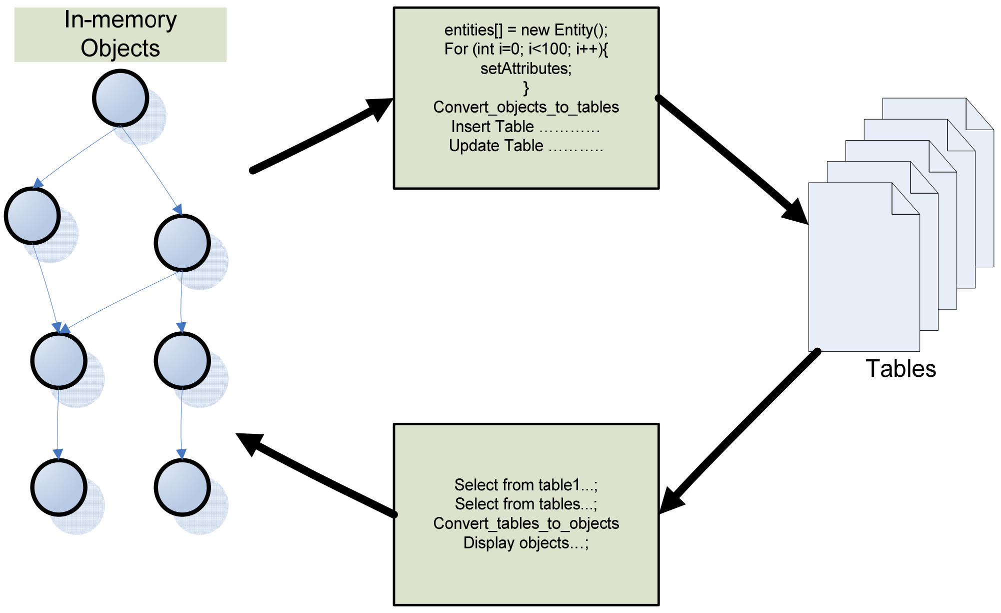


Figure 2.3: Impedance mismatch problem in Relational DBMSs.

(3) Relational DBMSs often use ER (Entity-Relationship) diagrams to model the static parts of an application and use another distinctive way to model operations and behaviors for entities in that application, which increases the difficulty for the modelling processes and prone to breaking the logical consistencies.

(4) Relational DBMSs are weak in "real word" representation (Lin, 2003 [17]). For example, relational DBMSs are difficult to represent "inheritance" and "aggregation" in the real word. Relational DBMS can only store data as entities. However, modern objects- and rules-based applications such as various knowledge-based systems for engineering, scientific (molecular biology) and multimedia applications often have specific operations (e.g. *setZResolution(String zResolution)* in the "*Instrument*" class category). These

data type specific operations are required to be encapsulated with the type rather than defining the stored procedures in separate way (Fu, 2002 [26]).

(5) Relational DBMSs are also incapable of supporting recursive mechanism and dealing with information inference (Hirao, 1990 [27]).

(6) As in commercial DBMS area, in order to support advanced applications, relational DBMS developed by Oracle, Microsoft, SUN and others attempted to incorporate some object-oriented features. Thus, for advanced applications supported by relational DBMSs, a large-sized management system has to be involved, which may cause prolonged running time and heavy system resources cost.

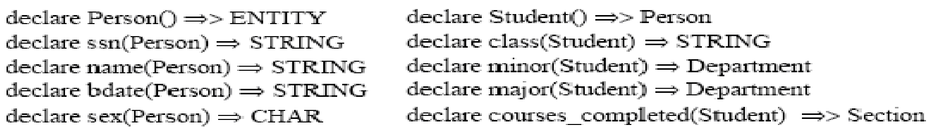
2.2.3 Object-relational DBMS

To help relational DBMSs to overcome problems highlighted in Section 2.2.2.4, various database vendors such as Oracle and SQLserver have devised the concept of "object-relational" DBMSs. The intention of object-relational DBMS is to integrate object-oriented features into relational DBMS, while still maintaining its relational DBMS background. There are no independent data models or standards used in object-relational DBMSs. Data models used in object-relational DBMSs extend the relational data model by providing additional inclusion of classes, inheritances and functions (e.g. query for complex data constructs) (Gray, 1992 [21]; Hirao, 1990 [27]). The functional data model is actually one of these extended data models. Therefore, this section focuses on discussing the functional data model and a DBMS base on it— P/FDM (Gray, 1992 [21]).

2.2.3.1 P/FDM Overview

The functional data model views a database as a collection of extensively defined functions that can be queried by functional query languages (Buneman, 1997 [28]). In the opinions of researchers attending the functional data model workshop in 1997, the functional data model is the "lowest common denominator" of data models, which can be seen as the basis for any other models used in relational, object-relational, and object-oriented databases (Buneman, 1997 [28]; Gray, 1997 [29]). Therefore, the functional data model can be used to explain the object-oriented concept. This sub-section will look in particular at the object-relational system — P/FDM, a research development by the Object Database Group at the University of Aberdeen (Embury, 1995 [30]). The P/FDM is based on functional data model having both a DAPLEX query interface and a query language in SICStus prolog (Intelligent Systems Laboratory, 2006 [31]). The basic features of the P/FDM are:

(1) In the P/FDM, database entities are represented as special functions that are devoid of parameters. Attributes in database entities are represented as functions that are applied to entities and return values of certain data types as results. Relationships between entities are also represented as functions that are applied to entities and return entities as results. Figure 2.4 shows examples of database entities, attributes and relationships defined in the functional database schema.



```
declare Person() =>> ENTITY            declare Student() =>> Person
declare ssn(Person) => STRING          declare class(Student) => STRING
declare name(Person) => STRING         declare minor(Student) => Department
declare bdate(Person) => STRING        declare major(Student) => Department
declare sex(Person) => CHAR            declare courses_completed(Student)  =>> Section
```

Figure 2.4: An example of functional database schema.

(2) Relationship functions in P/FDM can be reversed.

(3) Most Functional DBMSs are using DAPLEX language as a high-level query language. DAPLEX is a declarative language allowing non-experts to express what one wants without considering how the desired result is to be computed.

2.2.3.2 Advantages

The advantages of object-relational DBMSs can be summarized as:

(1) The data models in object-relational DBMSs keep the advantages of relational data model. For example, the functional data model is based upon functions and is a conceptual data model, which disconnects conceptual data model designs from those storage notions such as arrays, lists and other storage types (Gray, 1992 [21]; Gray, 1997 [29]).

(2) The extended data models in object-relational DBMSs provide better support for complex objects. For example, the functional data model integrates entities and behaviors in a unified model. It has capabilities for dealing with complex data structures such as potential embedded constructions and many-to-many relationships

(3) Object-relational DBMSs based on functional data model can have a function compositional query language, which can be recognized as a basis for the Object Query Language (OQL).

(4) Object-relational DBMSs based on functional data model have ability to integrate data from heterogeneous models in a multi-database scenario.

2.2.3.3 Disadvantages

The disadvantages of object-relational DBMSs can be summarized as:

(1) As the object-relational DBMSs are developing to offer virtually all the functionality currently required by object-oriented applications, they have become very large and cumbersome systems, which affect their usability.

(2) The Set Theory-based relational model has been extended in several ways that break the essential scope of Set Theory and without a comprehensive theoretical basis (Nelson, 1998 [32]).

(3) Data models for object-relational DBMSs still lack well-defined graphic notations and structures to represent complex structures. Complex objects and abstractions such as inheritance are more naturally represented by graphs than as sets (Levene and Poulovassilis, 1991 [33]; Poulovassilis and Levene, 1994 [34]).

(4) Data modellings in object-relational DBMSs still have weaknesses in semantic supports. For example, because the functional data model is the "lowest common denominator" of data models (refer to Section 2.2.3.1), there is lack of semantic foundations for knowledge-based applications. Compared to E-R model, the functional data model cannot clearly represent the stratification of entities, attributes, and relationships.

2.2.4 Object-oriented DBMS

To solve the problems presented by the aforementioned object-relational DBMSs, especially for these scientific and engineering applications, the next generation of database paradigm —object-oriented DBMSs are emerging. Several standards were proposed to develop object-oriented DBMSs, which include three manifestos, and standards from Object Data Management Group (ODMG93, ODMG2.0, ODMG3.0) as well as Object Management Group (OMG).

2.2.4.1 Object-oriented DBMS Standards

There are three manifestos for specifying the object-oriented DBMSs. The first manifesto provided 13 features that an object-oriented DBMS must include, should include and may include (Atkinson, 1990 [35]). However, there are no details and largely ignored the important part of object-oriented DBMSs − the data model. So it is just a vision not a formal standard (Committee for Advanced DBMS Function, 1990 [36]). The second manifesto detailed the first manifesto and proposed three tenets for the definition of the third generation DBMS: object-oriented DBMS. However, they focus too much on the evaluation of relational DBMSs to object-oriented DBMSs and ignored the intrinsic properties of object-oriented DBMS. It still absents the formal data model for object-oriented DBMSs. The third manifesto just follows and improves the earlier two manifests, and introduced how to add object-oriented features in relational DBMSs according to the different levels of object-oriented suggestions: object-oriented prescriptions, object-oriented proscription, and object-oriented very strong suggestions. These three so-called "object-oriented DBMS" manifestos are just guidance for extending relational DBMSs with object-oriented features, which are not real standards for object-oriented DBMSs.

The OMG is an international non-profit organization supported by information systems vendors, software developers and users. OMG was founded in 1989, now has over 600 member organizations, and meets bi-monthly. OMG provides a widely supported framework for open, distributed, interoperable, scalable, reusable, portable software components based on OMG-standard object-oriented interfaces (OMG, 1997 [37]). The object model of OMG provides minimum capabilities for object modelling.

From 1993, the ODMG developed a set of ODMG standards − ODMG93 in 1993, ODMG2.0 in 1997, ODMG 3.0 in 2000 (Cattell, et al., 1993 [38]; Cattell, et al., 1997 [39]; Cattell, et al., 2000 [20]). After ODMG3.0, the ODMG disbanded in 2001. The ODMG is a consortium of vendors and related organizations that work on standardization for object database and object-relational mapping products (Lin, 2003 [17]).The newest ODMG standard−the ODMG 3.0 defines a portability specification for persistent object storage, which enables portable applications that could run on more than one product. The ODMG3.0 binds object-oriented languages such as Java, C++ and Smalltalk, so application developers can entirely develop their database applications within the native language environment. Unlike to the aforementioned three manifestos, ODMG 3.0 is not developed on relational model, but directly built by scratching from object-oriented programming paradigms. It can also satisfy all core features defined in the first manifesto. The major components of ODMG 3.0 are (Cattell, 2000 [20]):

1. Object model. The object model defined in ODMG is developed based on the object model of OMG. The OMG object model is based on a small number of basic concepts: objects, operations, types, and subtypes (OMG, 1992 [40]), which is a common basis for Common Object Request Broker Architecture

(CORBA), object-oriented databases, and other object programming. ODMG refined the OMG's object model to specially satisfy the demands of object-oriented DBMSs by adding constructions such as relationships and transactions.

2. Object specification languages. Two specification languages defined in ODMG 3.0: Object Definition Language (ODL) and Object Interchange Format (OIF). The ODL is a specification language used to define the specifications of object types that conform to the ODMG object model. The ODL defined in ODMG is intended to define object types that can be implemented in a variety of programming languages, and it is not tied to the syntax of a particular programming language. The OIF is a specification language used to dump and load the current state of an object database to or from a file or set of files.
3. Object Query Language (OQL). The OQL is a declarative language for querying and updating objects developed on SQL-92.
4. Language Binding. The ODMG standard does not provide an Object Manipulation Language (OML) specification, instead ODMG provides language bindings. There are bindings of ODMG implementations to C++, Smalltalk, Java languages respectively. Due to the differences inherent in the object models native to these programming languages, it is not always possible to achieve consistent semantics across the different programming language-specific versions of ODL.

The ODMG Java language binding was the basis for Java Data Objects (JDO), an API for transparent persistence. The JDO API is a standard interface-based Java model abstraction of persistence, which developed under the auspices of the Java Community Process (SUN/JDO, 2008 [41]). The JDO focuses on standardizing the interfaces between host Java applications and databases. As the JDO has good portability, application programmers can focus on their domain object model, leaving the details of persistence (field-by-field storage of objects) to the JDO implementation. Therefore, JDO actually is the API working with both relational and object databases and it is not a database or a data model: it is a persistence API that can be used with a variety of data stores, including relational databases.

Besides standards discussed above, there are several other standards for defining object models, such as CFI (electrical CAD), PCTE (CASE), and ISO ODP (Lin, 2003 [17]). None of these standards have been widely accepted in the commercial market.

Hence, most of these standards are not specific to object-oriented DBMSs. The formal data model with mathematical foundation and a visualized diagramming mechanism is largely ignored in these standards. Nowadays, modern object-oriented programming languages enable object-oriented DBMSs to be developed more simply than before, so a data model that can ensure the integrity and consistency of stored objects becomes very important for object-oriented DBMS developments

2.2.4.2 Object-oriented DBMS Overview

In recent years from 1985, object-oriented DBMSs have received more attention and many experimental and theoretical practices have been done. An object-oriented DBMS integrates object-oriented features with database capabilities. It aims to address the limitations of relational databases by allowing complex data structures (objects and behaviors) to be stored in the database as objects. The object-oriented DBMS uses object-oriented programming languages to implement the attributes and behaviors of objects according to users' special demands and supports distributed applications. A number of object-oriented DBMSs come into markets in the past 15 years and Table 2.7 lists four object-oriented DBMS products that dominate today's market.

Vendors	Latest Products	Started
DB4O	Db4o (Database for Objects) v7.0	2004
The Ozone Database Project	Ozone v1.2	2002
Objectivity, Inc.	Objectivity/DB v9.0	1993
Versant Corporation	Versant v7.0	1988

Table 2.7: Current object-oriented DBMSs.

The Table 2.8 shows a comparison of the basic characteristics of object-oriented DBMSs listed in Table 2.7.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
User defined data types support	YES	YES	YES	YES
Inheritance (IS_A relationship) support	YES	YES	YES	YES
Aggregation (PART_OF relationship) Support	YES	YES	YES	YES
Version Support	YES	YES	YES	YES
The cardinality between objects check	NO	NO	YES	YES
Support of data replication	YES	NO	YES	YES

Data encryption support	YES	NO	YES	YES
Languages for defining attributes and methods of objects	JAVA, C#	JAVA	C++ JAVA SMALLTALK SQL	C, C++ JAVA SMALLTALK
Application programming in JAVA	YES	YES	YES	YES
Store methods of objects in the DB	NO, METHODS ARE STORED IN THE CLIENT	NO, METHODS ARE STORED IN THE CLIENT	NO, METHODS ARE STORED IN THE CLIENT	YES
Lock strategy	OBJECT LEVEL	OBJECT LEVEL	OBJECT LEVEL	CONTAINER LEVEL

Table 2.8: A Comparison of the basic characteristics for these four object-oriented DBMSs.

Table 2.9 shows a comparison of the standards supported by these four object-oriented DBMSs.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
ODMG ODL support	NO	Partially	Partially	NO
ODMG OQL support	NO	Partially	YES(supports all of SQL-92 which includes sql select with method execution, but not oql typing that differs from SQL-92)	NO
ODMG Java bindings	NO	YES	NO	YES(all basis capabilities (ref, relationships, etc.), but not collections
SQL query support	YES	YES	YES	YES

Table 2.9: A Comparison of the standards supported by these four object-oriented DBMSs.

Table 2.10 shows a comparison of the schema modification support of these four object-oriented DBMSs.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
Ad-hoc queries updates of the database schema with a GUI	YES	NO	YES	YES
Ad-hoc updates of the database schema with a object-oriented language	YES	YES	YES	YES

Table 2.10: A Comparison of the schema modification support for these four object-oriented DBMSs.

Table 2.11 shows a comparison of the queries support of four object-oriented DBMSs.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
Ad-hoc queries with GUI	YES	NO	YES	YES
Ad-hoc queries with SQL	YES	YES	YES	YES
Ad-hoc queries with a object-oriented language	YES	NO	YES	YES

Table 2.11: A Comparison of the queries support for these four object-oriented DBMSs.

Table 2.12 is a comparison of system environment support of these four object-oriented DBMSs.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
• **Architecture**				
Multi-user environment support	YES	YES	YES	YES
Single-user multi-tasking environment support	YES	YES	YES	YES
Client-server architecture support	YES	YES	YES	YES
The physical data can reside on client side	YES	YES	YES	YES
The application can run autonomously on the client side	YES	YES	YES	YES
• **Platform**				
MS-windows support	YES	YES	YES	YES
Sun OS support	YES	YES	YES	YES

Table 2.12: A Comparison of system environment support for these four object-oriented DBMSs.

Table 2.13 is a comparison of library file sizes, weak reference cache support and prices of these four object-oriented DBMSs.

Products	Db4o(v7.0)	Ozone(v1.2)	Objectivity(v9.0)	Versant (v7.0)
Library file sizes	3.5M	14M	165.7M(with objectivity/assist perspective)	YES
Weak Reference Cache	YES	NO	YES	NO
Price	Open Source	Open Source	About 3000$ per seat	About 3500$ per seat

Table 2.13: A Comparison of the accessibility for these four object-oriented DBMSs.

The core features of the object-oriented are highlighted as:

(1) All information is represented in the form of "object" and objects are stored persistently (Bancilhon, 1992 [42]).

(2) Objects in object-oriented DBMSs have properties (attributes) and behaviors (methods), which can be regarded as instances of entities in real world. Each of the objects has a uniquely assigned Object Identity (OID) and objects allow inheriting and overriding by arbitrary levels (Connolly and Begg, 2001 [16]).

(3) Object-oriented DBMSs combine database principles (Atomicity, Consistency, Isolation, and Durability) with object-oriented programming language principles (Encapsulation, Inheritance, and Polymorphism).

(4) Object-oriented DBMSs have query languages for accessing information.

2.2.4.3 Advantages

The advantages of using object-oriented DBMSs can be summarized as:

(1) Object-oriented DBMSs can deal with arbitrary complexity of object structures and object relationships of the real world in an efficient way — objects in an object-oriented DBMSs can hold arbitrary number of data of any types or even as other objects. Moreover, object-oriented DBMSs can use multi-valued properties to express complex data structures whilst in the relational model it can only be achieved by using additional relations and joins.

(2) Object-oriented DBMSs can encapsulate objects and their behaviors as an integrated whole, which is great benefit for certain types of applications. For example, multimedia applications use operations stored with objects to ensure the correct interpretation of special data (Bancilhon, 1992 [42]). It also means that the object-oriented DBMSs can use one data model to handle both static (entities and relationships) and dynamic (behaviors) aspects of an application.

(3) There are no impedance mismatch problems in object-oriented DBMSs applications. The conversions between object-oriented programming languages and databases (objects to table tuples) are not required since objects are using a uniform format in both object-oriented programming languages and databases, hence reducing the time cost for the unification tasks relating to transfer objects into tuples and vice versa (Obasanjo, 2001 [43]). Figure 2.5 shows an improved model in comparison with the model in the Figure 2.3:

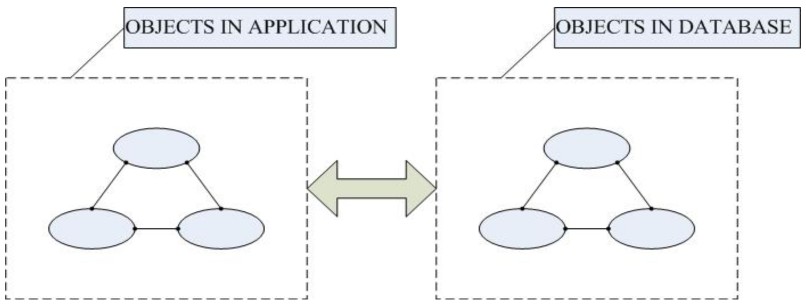


Figure 2.5: Unification between object-oriented application and object-oriented DBMS.

(4) Object-oriented DBMSs can automatically assign a unique ID for each object, which cannot be modified by applications and is independent of how an object is manipulated or structured (Bagui, 2003 [44]). This feature has avoided the daunting tasks facing by database designers in terms of defining keys. Most object-oriented DBMSs can convert the ID stored in an object to memory pointer when the object is loaded into the memory, therefore objects can be retrieved directly. Furthermore, two objects with different IDs will be considered as different objects even if their structures and property values are the same.

(5) Object-oriented DBMSs can naturally use class inheritance concept to model the hierarchy structures in real applications.

(6) Rather than using joins through primary-foreign key matches between tables, object-oriented DBMSs use object direct reference (e.g. reference pointer).

(7) Data access can be faster in object-oriented DBMSs than relational DBMSs since object-oriented DBMSs do not needs to search through tables using the time consuming join operations as in relational DBMSs. Furthermore, there are no needs to involve Call Level Interfaces (CLI) such as ODBC, ADO, and JDBC (Bagui, 2003 [44]).

(8) Additional query languages are not necessary for object-oriented DBMSs. The object-oriented programming languages such as Java, C# could be used to express queries − Native Queries (NQ).

2.2.4.4 Disadvantages

The disadvantages existing in object-oriented DBMS are:

(1) At present, object-oriented DBMSs are still lack of universal agreed standards and a formal basis to ensure the database systems remain a coherent and reliable system as new knowledge is being added and vendors of object-

oriented DBMSs are all relatively small (McClure, 1997 [45]).

(2) Object-oriented DBMSs are lack of a formal mathematical foundation, which leads to weaknesses in query and updating supports of object-oriented DBMSs.

(3) A contradiction exists between the encapsulation requirement of object-oriented DBMSs programming languages and database natures. For example, the query results are often formed by data values which will break the encapsulation rules. They are difficult or impossible to be stored back to object-oriented DBMSs or used in further queries.

(4) Difficulties exist when database schema changes (Obasanjo, 2001 [43]). In traditional relational DBMSs, the schema updating operations such as creation, deletion and modification of tables are actually independent with the host application. However, in an object-oriented DBMS based application, modifying database schema using similar operations on a persistent class may cause changes on other classes referring or interacting with the old instances of the class.

(5) There is a lack of portability. Object-oriented DBMSs are application specific, which are especially suitable for specific applications with specific purposes such as image processing, biological analysis, engineering standards handling, and physics applications. They are not particularly appealing to mainstream commercial applications.

(6) Due to lack of advanced features such as query facilities, query optimizations, view supporting, security issues and consistency checking, object-oriented DBMSs do not have the maturity as relational DBMSs (Bagui, 2003 [44]).

2.3 The Survey of Knowledge-based System

2.3.1 Definition of a Knowledge-based System

There are various definitions of a knowledge-based systems defined by various authors, researchers and software system experts. In many papers or books, a number of authors imply that an expert system and a knowledge-based system are equivalent since knowledge-based systems are used to capture the problem-solving expertise of human beings, which is closing to the generally accepted definition of expert systems. In 1989, Mockler gives a definition for a knowledge-based system as "designed to replace the functions performed by a human expert". Dym and Levitt in 1991 explicitly make no distinction between a knowledge-based system and an expert system. They define a knowledge-based/expert system as “a computer program that performs a task normally done by an expert or consultant and which, in so doing, uses captured, heuristic knowledge” (Dym and Levitt, 1991 [46]). On the other hand, there are also definitions that make an explicit distinction between a knowledge-based system and an expert system. In this project, researchers make an expert system is a subset of a knowledge-based system, which means that the expert systems have more advanced inferences than knowledge-based systems in solving decision-making problems. Therefore, a definition that focuses on the knowledge carrying in systems rather than broader or advanced inference powers is used in this project (Harmon and King, 1985 [47]):

“*Today’s knowledge systems are confined to well-circumscribed tasks. They are not able to reason broadly over a field of expertise. They cannot reason from axioms or general theories. They do not learn and, thus, they are limited to using the specific facts and heuristics that they were ‘taught’ by a human expert. They lack common sense, they cannot reason by analogy, and their performance deteriorates rapidly when problems extend beyond the narrow task that they were designed to perform.*”

Figure 2.6 illustrates the relationship of the knowledge-based system, expert system and artificial intelligent applications in a tree structure (Partridge and Hussain, 1995 [7]).

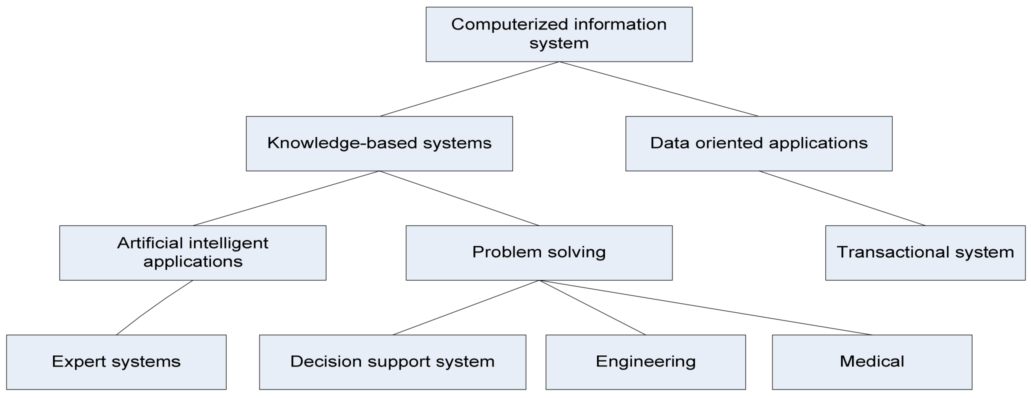


Figure 2.6: A Classification of information systems.

2.3.2 Knowledge-based System Overview

Knowledge-based systems originated in 1943, and have evolved during the last sixty years in several branches: artificial intelligence applications, expert systems, decision support systems and so on. Table 2.14 lists the milestones related directly to the development of knowledge-based systems (Partridge and Hussain, 1995 [7]).

Years	Milestones
1943	McCulloch and Pitts published the pioneering work that led to neural networks.
1950	Alan Turing's article that led to the Turing test.
1956	Newell, Simon, and Shaw developed the general problem solver at Rand Corporation.
1958	McCarthy developed LISP at Massachusetts Institute of Technology (MIT).
1963	Samuel's article on the first learning computer program (Samuel's draughts program) and in the use of techniques of search and reasoning.
1970	Rousel and Colmerauer developed PROLOG.
1972	Newell and Simon's book "Human Problem Solving" introduced the general idea of production systems.
1973	Van Melle, Shortliffe, and Buchanan developed the EMYCIN shell from MYCIN
1976	Minsky developed the concept of frames for knowledge representation
1977	Forgy created OPS for programming expert systems.
1978	McDermott started developing R1 (later released as XCON, the first large commercial expert system) at Digital Corporation.
1980	Symbolics started the development of LISP machines.

Table 2.14: The development milestones for knowledge-based systems.

In real world applications, there are many kinds of knowledge-based systems, which mixed by different kinds of knowledge sources, see Table 2.15 (Partridge and Hussain, 1995 [7]).

Knowledge Types	Characteristics	Output	Attributes	Relationship of knowledge to problem solving
Facts	Statement of existence	What is	Truth	Data
Heuristics	Rule of thumb	Why and why not	Discovery	Tactics
Rules	Relationship of factual conditions and conclusions	What should be	Conditions associated with actions and conclusions	Tactics
Procedure	How things work	How it is done	Algorithms	Procedure
Declarative(descriptive)	How things are	Why it is done	Association with truth	Strategies

Table 2.15: Knowledge types.

These five kinds of knowledge are all mixed in the VirtualGPS system and Table 2.16 lists examples.

Knowledge Types	Examples in VirtualGPS system
Facts	Parameter types and tolerance values
Heuristics	Patterns in pattern language for function reports
Rules	Constraints between symbols in a completed callout (e.g. between parameter types and sampling lengths)
Procedures	Comparison processes in Verification
Declaratives	Manufacture reports for suggesting manufacture strategies(manufacturing processes and tools)

Table 2.16: Knowledge types in VirtualGPS.

The next section gives an introduction into the hierarchy of knowledge and their relationships.

2.3.3 Knowledge Hierarchy

According to Russell Ackoff, a system theorist and professor of organizational changes, the content of the human mind can be classified into five categories (Ackoff, 1989 [48]):

(1) **Data**. Data is a symbol set, which is raw and has no significance beyond its existence. In the computing world, data is records, signals or other encoded items.

(2) **Information**. Information is data that has been given meaning by way of relational connection. This “meaning” can be useful, but does not to be.

(3) **Knowledge**. The knowledge is application of data and information to answer “how” questions. Knowledge must have useful meaning.

(4) **Understanding**. Understanding is an interpolative and probabilistic process, which human beings can take knowledge and synthesize new knowledge from the previously held knowledge. The difference between understanding and knowledge is the difference between the “learning” and “Memorizing”. Artificial Intelligent systems process understanding in the sense that they are able to synthesize new knowledge from previously stored information and knowledge.

(5) **Wisdom**. The wisdom is an extrapolative and non-deterministic, non-probabilistic process, which is used to evaluate understandings to make judgments on understandings.

Figure 2.7 illustrates the hierarchy of data, information and knowledge.

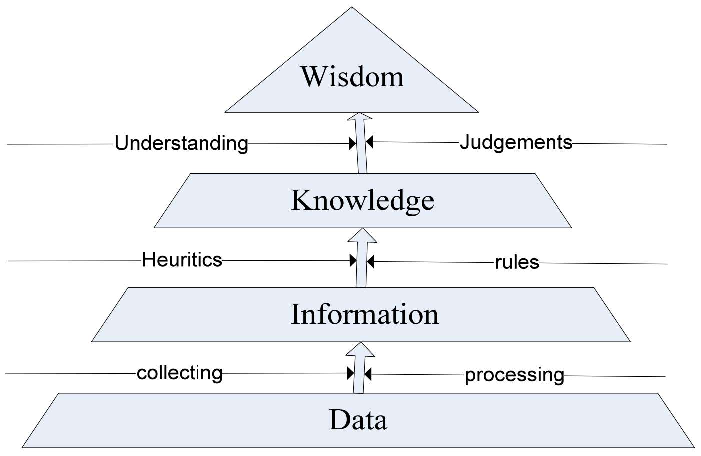


Figure 2.7: The hierarchy of knowledge.

At present, this project not aim to yield wisdom by judging existing knowledge, but the VirtualGPS focus on representing and inferring knowledge based on rules and cases retrieved from GPS standards.

2.3.4 Knowledge Acquisition and Representation

Figure 2.8 shows a classic architecture for a knowledge-based system (Partridge and Hussain, 1995 [7]; Hopgood, 2001 [49]).

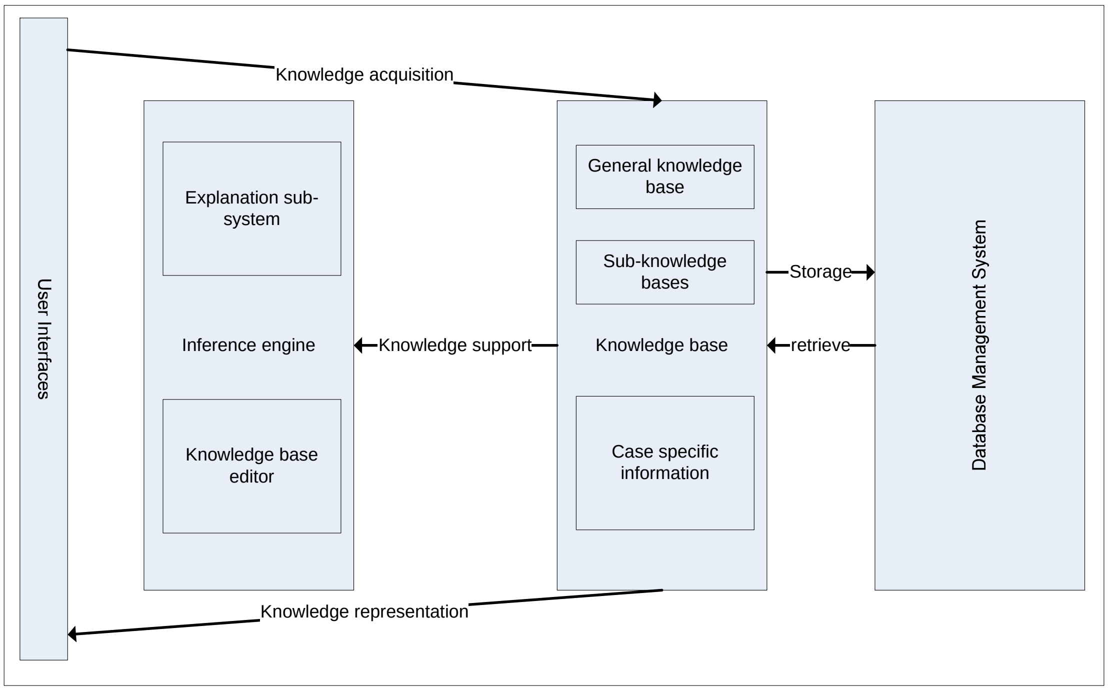


Figure 2.8: The classic architecture for knowledge-based systems.

According to Figure 2.8, knowledge acquisition and knowledge representation are the first two steps which need to be gone through during the developing process of a knowledge-based system. Knowledge acquisition is the first step in creating a knowledge base. There are three distinct approaches to acquiring the relevant knowledge for a particular domain (Hopgood, 2001 [49]):

- The knowledge is teased out of a domain expert.
- The builder of the knowledge-based system is a domain expert.
- The system learns automatically from examples.

The first approach is commonly used for knowledge acquisitions, but has a major problem: misunderstandings between knowledge system developers and domain experts. The communication difficulties can be avoided or alleviated through defining a clear knowledge representation mechanism. Knowledge bases require special representations for knowledge. After evaluating common data models such as hierarchical, network, and relational that are used in DBMSs, Partridge and Hussain claimed that these traditional data models for DBMSs are not adequate for AI applications where knowledge is used to make inferences (Partridge and Hussain, 1995 [7]). Therefore, knowledge base developers developed some special data representations for knowledge like rules, the frame, the semantic network, logic, and the object-oriented approach. This leads to a gap between data in the database and data in the knowledge base. Actually, the knowledge representation and knowledge bases do not replace data representation and databases. Instead, they are all necessary parts for a knowledge-based system. The DBMS is used to store and manage data for knowledge-based system while knowledge base uses data stored in DBMS to organize information or knowledge for users. Hence, researchers in this project devised a categorical object model that can be used for both in knowledge base for knowledge representation, and in a DBMS for object and complex relationship modelling. This project also devised representation mechanism for reasonings based on Category Theory (coequalizer) that can be used to control the logical manipulations of objects to generate new knowledge from old (see Section 3.6 of Chapter 3). Another example on using Category Theory to represent heuristics and theories can be seen in Section 3.7.1 of Chapter 3. Furthermore, while using the categorical object model for representing inference rules or constraints in a diagrammatical way with certain level of abstraction, the traditional knowledge representation mechanisms− rule and frame are also used in this project to specify rule contexts and heuristics in detail. The rule is a knowledge representation for making inferences as a human expert does, which is applied to knowledge to get a conclusion or activate an action (Partridge and Hussain, 1995 [7]). The basic format for a rule is (Tansley and Hayball, 1993 [50]):

IF x THEN 1.
ELSEIF y THEN 2.

List 2.1: The basic format for a rule.

Chapter 5 demonstrates a set of rules in the VirtualGPS system. These rules are held in the knowledge base of this system to inference knowledge from existing knowledge stored in the categorical DBMS. The frame is a data structure for representing a stereotyped situation, which contains a set of slots and nodes organized in logic groups (Partridge and Hussain, 1995 [7]). Slots containing rules, values, pointers to other frames and procedures are used to define an event or a concept at each node. A node is a point where an item links to another item. Frames are very useful when the content of information is important in solving problems containing patterns. The pattern language (see Section 3.7.1 of Chapter 3) and PRIMA (Manufacturing Process Information Map) (see Section 5.4.2 of Chapter 5) are two kinds of frames in this project and they are formatted by using Category Theory.

This section shows that the Category Theory provides a high unification and abstraction for knowledge acquisition and knowledge representation for this project to eliminate knowledge design complexity and communication misunderstandings between knowledge designers and system developers.

2.4 XML/XSLT

2.4.1 XML Definition

Extensible Markup Language (XML) is

"*A technology for making up structured data so that any software with an XML parser can understand and use its content. Data independence, the separation of content from its presentation is the essential characteristic of XML*" (Deitel et al., 2003 [51]). Like any other markup languages, XML has a set of rules, which the user can use to add special meanings or provide extra information to a document. However, unlike HTML, tags used in XML are not pre-defined, users can define their own tags to make up data and these user-defined tags only relate to the actual content of the document, not the way to display it. Therefore, "HTML is used to define how a document should be rendered, whereas XML is used to define the data contained within that document" (Reynolds, 2000 [52]). Because XML based on user-defined tags, the browser will not know how to display an XML document, so users often use Extensible Stylesheet Language (XSL) to tell the browser how to display it. In order to make the XML document more readable, XML also use Document Type Definition (DTD) or an XML Schema to define the legal elements in an XML document.

Data often communicates between different platforms, systems, and applications in different formats. XML can define the content of a document separately from its formatting and presentation, making it easy for data communication between different platforms, systems and applications.

2.4.2 Advantages of XML

XML is a nice tool to exchange data. XML documents are simple text files marked up in a special way, so all applications can use XML data expediently. XML provides a basic syntax that can share information between different kinds of computers, applications, and organizations without needing to pass through many layers of conversion. XML is not complex to use. It has a set of syntax rules, which protect a developer to build a correct XML document. XML has very good extensibility. The extensibility of XML shows in two ways: user can use DTDs to define rules for their own tags and XML can support many other standards such as XSL, XPointer, CSS, Xlink. Finally, XML is completely open, freely available on the web.

2.4.3 XSLT

The Extensible Stylesheet Language (XSL) provides rules for formatting XML documents (Deitel et al., 2003 [51]). XSL Transformation Language (XSLT) is a core part of XSL, which can transforms an XML to other text-based forms such as XHTML pages, WML cards or PDF files.

2.4.4 DOM

An XML document is represented by a hieratical tree structure in memory. This structure includes the elements, attributes and content of the document (Deitel et al., 2003 [51]). Document Object Model (DOM) was developed by W3C, which can dynamically build a hierarchical tree in memory for a XML document and each node in DOM tree represents an element, attribute or content of a XML document.

2.5 Summary

This chapter gives a brief overview and discussion on the problem domains relating to this project. It also illustrates why this project needs to be done and why new techniques need to be involved.

CHAPTER 3 CATEGORY THEORY APPLICATIONS

This chapter focuses on discussing the Category Theory modelling capability used in this project. It covers: introduction of necessary notations and constructs of Category Theory used in this project, discussion of related previous researches of Category Theory, justification of the rationales for choosing Category Theory in this project, and explanations for the categorical modelling mechanism. The categorical modelling mechanism devised in this project contains three parts: the categorical object model, the categorical software design process and the inference identifying square, which covers all aspects of object-oriented knowledge-based system modellings.

3.1 Category Theory

Category Theory is a form of constructive mathematics, which is devised to describe various structural concepts from different mathematical areas in a uniform foundation. As claimed by Goguen in 1991, Category Theory can contribute the major six points for the modern computing science (Goguen, 1991 [53]):

- Formulating definitions and theories. The Category Theory provides a symbolic language with a convenient symbolism that allows for visualization of quite complex facts in form of diagrams (Adamek et al., 1990 [54]).
- Dealing with abstraction and representation independence. Category Theory can grasp the essence of the researching targets as it focuses on the properties of mathematical structures instead of on their detail representations. For example, the diagram in Category Theory is similar to the graph in Topologic Theory, which is used to model pairwise relations between objects in a certain domain instead of focusing too much on precise positions of those objects.
- Carrying out proofs. By using diagram chasing and calculus deductions, Category Theory can reduce all complex proofs to simple calculations.
- Discovering and exploring relations with other fields. Sufficiently abstract formulations can reveal surprising connections. For example, an analogy between Petri nets and the λ-calculus might suggest looking for a closed category structure on the category of Petri nets (Meseguer and Montanari, 1988 [55])
- Formulating conjectures and research directions. Connections with other fields can suggest new questions in your own field. For example, if a special functor

has been found, the investigations of its adjoints can be very valuable.

- Unification. Computing science is very fragmented with many different subdisciplines, so the Category Theory can provide a conceptual unification.

3.1.1 Notations of Category Theory

Category Theory has two basic notations − category and arrow. A category ***C*** is for specifying complex structures and formalisms, which can contain (Barr and Wells, 1996 [56]; Pierce 1991 [57]).

- A collection of internal objects (More specially, if an internal object I in ***C*** has a unique arrow to every other internal object of ***C***, it is defined as initial object and denoted as 0. The dual notion is the terminal object T denoted as 1 that there has exactly one arrow $X \rightarrow T$ for each object X of C).
- A collection of arrows/morphisms (e.g. f: $A \rightarrow B$). An arrow in Category Theory is similar to a function in Set Theory, which defines a mapping from a source to a target internal object. Functions or behaviors assigning to each arrow f with an object *dom(f)* (domain) and an object *cod(f)* (codomain) (e.g. f: $A \rightarrow B$, $dom(f) = A$, $cod(f) = B$), the collection of all arrows with domain A and codomain B in category ***C*** is represented as $Hom_C(A,B)$.
- a composition operator on each pair of arrows f and g satisfying $cod(f) = dom(g)$ (a composite $g \circ f$: $dom(f) \rightarrow cod(g)$);
- satisfying the associative law: for any arrows f: $A \rightarrow B$, g: $B \rightarrow C$ and h: $C \rightarrow D$ has $h \circ (g \circ f) = (h \circ g) \circ f$;
- an identity arrow id_A: $A \rightarrow A$, for each object A satisfying the identity law as for any arrow f: $A \rightarrow B$, $id_B \circ f = f$ and $f \circ id_A = f$;

Figure 3.1 shows a basic category:

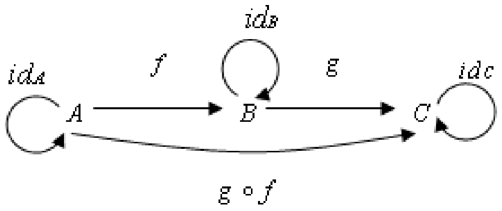

$$\begin{tikzcd} A \arrow[loop left, "id_A"] \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[loop above, "id_B"] \arrow[r, "g"] & C \arrow[loop right, "id_C"] \end{tikzcd}$$

Figure 3.1: A basic category.

For an instance, the ***Set*** category is a category where the internal objects are sets and the arrows are total functions. The subcategory ***S*** of a category ***C*** is a category that every internal object of ***S*** is an internal object of ***C***; for all objects O and O' in ***S***,

$Hom_S\,(O, O') \subseteq Hom_C\,(O, O')$.

Arrows in Category Theory have three important types: monomorphisms, epimorphisms, and isomorphisms (Barr and Wells, 1996 [56]). The definition for monomorphism (as shown in Figure 3.2) in Category Theory is:

"*An arrow f: A → B is a monomorphism (also called a monic morphism or a mono), if for any object C of the category and any arrows* $g_1,\ g_2\ : C \rightarrow A$ *such that* $f \circ\ g_1 = f \circ\ g_2$ *implies* $g_1\ =\ g_2$."

$$\begin{tikzcd} C \arrow[r, shift left, "g_2"] \arrow[r, shift right, "g_1"'] & A \arrow[r, "f"] & B \end{tikzcd}$$

Figure 3.2: Diagram representation of the definition of monomorphism.

The monomorphism in ***Set*** category corresponds to the concept of injective function. The definition for epimorphism (as shown in Figure 3.3) in Category Theory is:

"*An arrow f: B → A is an epimorphism (also called an epic morphism or an epi), if for any object C of the category and any arrows* $g_1,\ g_2 : A \rightarrow C$ *such that* $g_1\ \circ f = g_2\ \circ f$ *implies* $g_1\ =\ g_2$."

$$\begin{tikzcd} B \arrow[r, "f"] & A \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] & C \end{tikzcd}$$

Figure 3.3: Diagram representation of the definition of epimorphism.

The epimorphism in ***Set*** category corresponds to the concept of surjective function. An epimorphism is a monomorphism in the dual category. The dual category $\boldsymbol{C^{op}}$ of category ***C*** contains all internal objects same as ***C*** and all arrows of ***C*** inverted. The inverted arrow means, given an arrow *f*: *A* → *B* then the inverted arrow f^{op} of *f* is an arrow f^{op} : *B* → *A*. The notion of duality in Category Theory is very useful as it reduces proof obligations: the dual of a theorem is also a theorem.

The definition for isomorphism in Category Theory is:

"An isomorphism is arrow f: A → B if there exists f^{op} *: B → A, such that* $f^{op} \circ f = id_A$ *and* $f \circ\ f^{op} = id_B$. *The objects A and B are isomorphic if there is an isomorphism between them."*

The isomorphism in ***Set*** category is corresponding to the concept of bijective function.

Category Theory also provides several high-level concepts based upon the above basic ideas for the "category of categories" scenarios. There are four high-level concepts which give the multi-level mathematical capability considered relevant to this project:

- A high-level concept that is a special type of structure preserving mapping (arrow) between categories named as "functor". The formal definition of a functor is (Barr and Wells, 1996 [56]):

  "*Let **C** and **D** be categories. A functor F: **C**→**D** is a map taking each object A in **C** to an object F(A) in **D** and each arrow f:A→B in **C** to a arrow F(f):F(A) →F(B) in **D** while holding the following two properties:*
  - $F(id_A) = id_{F(A)}$
  - *F(g ∘ f) = F(g) ∘ F(f) for all arrows f:A→B and g:B→C.*"
- Functors again can be considered as categories (functorial categories), so an arrow between functorial categories is the "natural transformation" as shown in Figure 3.4.

  "*If F and G are covariant functors between the categories **A** and B, then a natural transformation η from F to G associates to every object X in **A** a arrow : F(X) → G(X) in B called the component of η at X, such that for every arrow f : X → Y in **A** the following diagram commutes as η_Y ∘ F(f) = G(f) ∘ η_X*" (Saunders, 1998 [58]).

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] \arrow[d, "\eta_X"'] & F(Y) \arrow[d, "\eta_Y"] \\
G(X) \arrow[r, "G(f)"] & G(Y)
\end{tikzcd}$$

Figure 3.4: Commutative diagram for covariant natural transformation.

  Thus, the Natural transformation provides a way for transforming between functors while respecting the internal structure of the categories involved (Saunders, 1998 [58]). The covariant functors indicate that the domain *F(X)*, *G(X)* must have same type and codomain *F(Y)* and *G(Y)* must have same type. This is used to ensure the comparison mapping of natural transformation is meaningful. Thus, natural transformations used in this project are isomorphic, which map between functors in same structure.
- Category Theory uses the concept of "diagram" to represent complex

structures in a world scenario. Before giving the definition of diagram, the definition of "graph" should be given first as follows (Rydeheard and Burstall, 2003 [59]):

"*A graph is a pair N, E of sets (of nodes and edges) together with a pair of mappings s, t: E → N called source and target respectively. The f: a b represents when f is in E and s(f) = a and t(f) = b. A finite graph is one in which N and E are finite sets.*"

Based on the above definition, a definition of diagram can be defined as (Rydeheard and Burstall, 2003 [59]):

"*A diagram in a category **C** is a graph (N, E, s, t) (its shape) and two functions f : N →Obj(**C**), g : E→Arrow(**C**) which respect sources and targets in the following sense: For each edge e*$\in$*E, f(s(e)) =* s_A *(g(e)) and f(t(e)) =* t_A *(g(e)), where* s_A *and* t_A *are source and target objects of arrows in **C**.*"

After analyzing the above definitions, it is clear that the diagram in categorical view is a similar concept to the indexed family in the Set theory that can be treated as a functor *D*: ***T***→***C*** where category ***T*** is the index category and the diagram *D* is indexing a collection of objects and arrows (morphisms) in ***C*** using pattern ***T***. A diagram is said to "commute" if every path between two objects in its image can be determined through composition of the same arrow.

- The notion cone can be defined as:

  "*let D: **T**→**C** be a diagram in **C** and N be an object of **C**, thus a cone from N to D is a set of arrows (morphisms) —*ψ_X *:N→D(X) and for each object X of T such that for every arrow f: X→Y there has D(f)* $\circ$ $\psi_X = \psi_Y$ *.*"

  as Figure 3.5 demonstrating (Saunders, 1998 [58]).

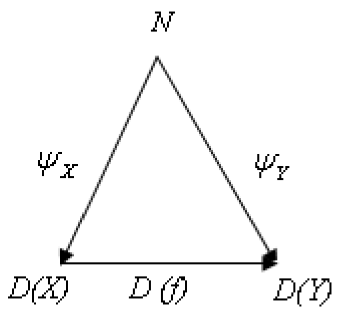
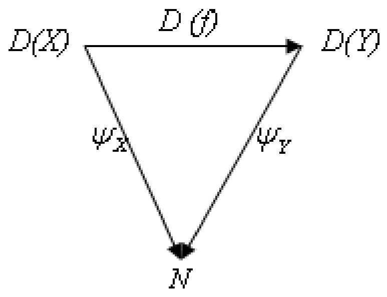

$$\begin{tikzcd} & N \arrow[dl, "\psi_X"'] \arrow[dr, "\psi_Y"] & \\ D(X) \arrow[rr, "D(f)"'] & & D(Y) \end{tikzcd}$$

Figure 3.5: Commutative cone.

Therefore, a dual notation of cone is cocone, as shown in Figure 3.6.



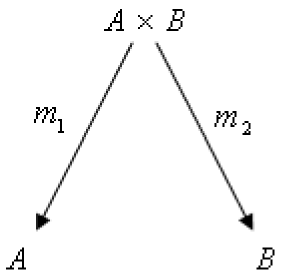
$$\begin{tikzcd}
D(X) \arrow[rr, "D(f)"] \arrow[dr, "\psi_X"'] & & D(Y) \arrow[dl, "\psi_Y"] \\
& N &
\end{tikzcd}$$

Figure 3.6: Commutative cocone.

3.1.2 Constructs

Based on the notations discussed above, a set of fundamental constructs has been formed, and will be used in this project.

(1) **Product**. A "product" in Category Theory can be diagrammatically illustrated by "cone" shown in Figure 3.7.



$$\begin{tikzcd}
& A \times B \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
A & & B
\end{tikzcd}$$

Figure 3.7: Product diagram.

A and *B* are internal objects in a category and *A* × *B* is also an object formed by *A* and *B* with specific relationships. The m_1 and m_2 are called as coordinate projections or simply projections which are functions: m_1 : *A* × *B* → *A* and m_2 : *A* × *B* → *B*. A formal definition of a categorical product is:

"*The product of two objects A and B is an object U, together with two projection arrows* m_1 *: U → A and* m_2 *: U → B, such that for any object C and pair of arrows f: C → A and g: C → B there is an exactly one mediating arrow <f, g>: C → U making the commute – that is, such* m_1 ∘ *<f, g> = f and* m_2 ∘ *<f, g> =g*" (Pierce, 1991 [57]).

In the ***Set*** category, products are in correspondence to the notion of cartesian products. See Figure 3.8.

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, dashed, "<f, g>"] \arrow[dr, "g"] & \\
A & U \arrow[l, "m_1"'] \arrow[r, "m_2"] & B
\end{tikzcd}$$

Figure 3.8: Commutative product diagram for objects *A* and *B*.

In Figure 3.8, *U* is the universal product of *A* × *B*. A pullback is a product with restricted objects (See Section 3.7.2). The product construct can also be applied to arrows:

"*A product of two arrows f:* A→*A' and g: B→B' is an arrow f* × *g : A* ×*B* → *A'* × *B' such that the following diagram commutes:"* (Guo, 2002 [60]).

$$\begin{tikzcd}
A \arrow[d, "f"] & A\times B \arrow[l, "\pi_A"'] \arrow[r, "\pi_B"] \arrow[d, "f\times g"] & B \arrow[d, "g"] \\
A' & A'\times B' \arrow[l, "\pi_{A'}"'] \arrow[r, "\pi_{B'}"] & B'
\end{tikzcd}$$

Figure 3.9: Commutative product diagram for two arrows.

The concept of diagram commutative is of vital importance for researchers to prove proofs and definitions and express equations.

(2) **Coproduct**. The construct of a "coproduct" in Category Theory can be diagrammatically illustrated by "cocone" in Figure 3.10.

$$\begin{tikzcd}
 & A+B & \\
A \arrow[ur, "n_1"] & & B \arrow[ul, "n_2"']
\end{tikzcd}$$

Figure 3.10: Coproduct diagram for object *A* and *B*.

The dual notion of a product is coproduct and its formal definition is:

"*A coproduct of two objects A and B is an object A* + *B, together with two injection arrows* n_1*: A* → *A* + *B and* n_2*: B* → *A* + *B such that for any object C and pair of arrows f: A* → *C and g: B* → *C there is exactly one mediating arrow [f,g]: A* + *B* → *C making the diagram commute – that is, such* $n_1 \circ [f, g] = f$ *and* $n_2 \circ [f, g] = g$" (Pierce，1991 [57]).

In the ***Set*** category, a coproduct corresponds to the notion of disjoint union. See Figure 3.11.

$$\begin{tikzcd}
A \arrow[r, "n_1"] \arrow[dr, "f"'] & A+B \arrow[d, dashed, "{[f,g]}"] & B \arrow[l, "n_2"'] \arrow[dl, "g"] \\
 & C &
\end{tikzcd}$$

Figure 3.11: Commutative coproduct diagram for objects *A* and *B*.

This definition for coproduct of objects can also be extended to arrows:

"*A coproduct of two arrows f: A→A' and g: B→B' is an arrow f + g : A' + B'→ A + B such that the following diagram commutes:*" (Guo, 2002 [60]).

$$\begin{tikzcd}
A \arrow[r, "I_A"] & A+B & B \arrow[l, "I_B"'] \\
A' \arrow[u, "f"] \arrow[r, "I_{A'}"] & A'+B' \arrow[u, "f+g"] & B' \arrow[u, "g"] \arrow[l, "I_{B'}"']
\end{tikzcd}$$

Figure 3.12: Commutative coproduct diagram for two arrows.

(3) **Limit and colimit**. "limits" and "colimits" are universal cones/cocones. The formal definition of limit is (Barr and Wells, 1996 [56]):

"*A limit for a diagram D is a cone* Ψ_X *: N→D(X) with the property that if* φ_X *: L→D(X) is another cone for D then there is a unique arrow v: L→N such that the following diagram commutes for every object X in D.*"

$$\begin{tikzcd}
L \arrow[rr, "v"] \arrow[dr, "\varphi_X"'] & & N \arrow[dl, "\Psi_X"] \\
 & D(X) &
\end{tikzcd}$$

3.13: A Limit for a diagram D.

Figure 3.13 provides a simplified illustration of a limit. The colimit can be defined as dual notion of a limit. If treating a diagram *D* with limit as a category, thus a limit is an initial object. In similar way, a colimit is a terminal object of a category that is a diagram with colimit. If a category ***C*** has an initial object, then it is unique up to isomophism and same true for its terminal object.

With a limit construct, the finite complete category can be formed when all finite limits exist (i.e. limits of diagrams indexed by a finite category). Dually, a category is finitely cocomplete if all finite colimits exist.

The concepts and constructs introduced above demonstrate that Category Theory has rich set of mathematical notions for both diagrammatic and algebraic theories. These notions can naturally model object-oriented applications, and are especially good at modelling multi-level architectures.

3.2 Pilot Researches into Category Theory

Category Theory originally rose in mathematics and was defined as an abstract way to deal with mathematical structures and relationships between them. It offers a formal basis and abstraction for handling the passage from one type of mathematical structure to another through mappings that preserve structures (Barr and Wells, 1996 [56]). It is still a maturing mathematical subject, which first emerged in 1945 in Eilenberg & MacLane's paper entitled "General Theory of Natural Equivalences" (Eilenberg and MacLane, 1945 [61]). In last three decades, Category Theory has found new applications in the theoretical computer science, algebra and database applications attributing to its firm mathematical roots, which contributed, among other things, to the development of semantic programming and new logical systems. In the literature, several papers has been published on the studies of Category Theory in computer science area such as database applications, software engineering, semantic algebra, information flow, etc. A short overview on previous researches relating this project is discussed in following paragraphs.

In 1985, Cartmell first used the categorical logic in database and then later in 1987, Ehrich, et.al., discussed using coproducts to model aggregation (Cartmell, 1985 [62]; Ehrich et al., 1987 [63]).

Goguen published a categorical manifesto in 1989, which focuses on discussing why and how the Category Theory is useful in computing science especially for expressing programming semantics (Goguen, 1989 [53]). This paper also gave guidelines for applying seven basic category notions: category, functor, natural transformation, limit, adjoint, colimit and comma category with some examples.

In 1990, a manifesto for categorizing database theory published by Cadish and Diskin gave proofs that the Category Theory can be naturally incorporated into object-oriented database modelling (Kadish and Diskin, 1997 [64]). They highlighted the graphic, algebraic and polymorphic nature of Category Theory, which can give an algebraic graph-oriented formal language for specifying structure and dynamics of the world. However, this manifesto contains too many slogans and lacks real cases or examples.

In 1991, Lellahi and Spyratos devised a categorical data model supporting structured objects and inheritance using concepts of graph, category and diagram. The directed labeled graphs are used to represent the database schemas: using the node concept in graph to represent class concept in object-oriented database, so these nodes are structured; directed edges with labels to represent inter-relationships. Then every structured node will be mapping to the limit of a diagram that is actually a finite category with limit. The limit of a diagram and limit of universal cone concepts used in Lellahi and Spyratos's data model are very valuable for defining the class category notion in this thesis. However, the gold points − rich semantic constructs and multi-level mappings of Category Theory are largely ignored in Lellahi and Spyratos's data model. The category and diagram concepts are only used to populate the database.

David Nelson and Nick Rossiter (Nelson and Rossiter, 1994 [65]; Nelson et al., 1994 [66]) developed a semantic data model for object-relational DBMS in 1994, which is an extension of the functional data model based on the Category Theory. This research gave the further proofs that Category Theory can be gracefully used in the database area. A prototype has been built based on P/FDM system. However, because of the limits of P/FDM, this DMBS is weak in dealing with dynamic aspects since arrows and functors in the DBMS can only perform static relationships between internal objects or categories.

In 1996, ter Hofstede designed a conceptual data model using the Category Theory, which extended the Lellahi and Spyratos's work (Hofstede et al., 1996 [67]). This approach devised a type graph, and then populated it with category theoretic formalizations. The later process mapped the object types in the type graph onto objects in the instance category, with their edges turn into arrows of the category. This method actually used a type graph to define the conceptual data model, as well as using the Category Theory formalizations to handle semantics of the data model. However, the work only focused on building specialized formalisms based mainly on graph theory and did not make the full use of the Category Theory to build a uniform data model for real database applications.

In 2001, Colomb adopted "fibration" concept in Category Theory for data refinement for data models in information systems (Colomb, 2001 [68]). This project used his methods to ensure the consistency between initial abstract modelling diagrams and final implementable modelling diagrams (see Section 3.5).

In 2002, the Guo claimed "*in today's large systems, the variety of encountered interconnection relationships (such as implements, uses and extends) is very large, while the complexity of protocols for managing them can be very high*". In addition, there are three problems for current software designs caused by the failure of many current tools to recognize software component interconnection as a distinct design entity. The three problems are: discontinuity between architectural and implementation models; difficulties in application maintenance; and difficulties in component reuse (Guo, 2002 [60]). This paper also pointed out that one of the major reasons for this failure is the lack of expressive means for representing interdependencies or coordination protocols as distinct and separate entities. In order to solve this problem, Guo tried to use Category Theory to provide distinct construct for modelling of the software component dependencies. However, the paper is just the initial thoughts of the author, which focuses so much on introducing formalizations of Category Theory, so no real example can be found in this paper.

In 2005, Lu and her colleagues developed types for morphisms and got the typed category for the abstract description of knowledge and knowledge processing (Lu, 2005 [69]). The paper published by Lu proposed that the typed Category Theory can be a mathematical abstraction of a set of various knowledge representation mechanisms such as semantic networks of Quillian, conceptual graph of Sowa, entity relationship diagrams of Chen and Allen's time algebra. The typed Category Theory is proposed differing from traditional Category Theory in two aspects: all morphisms (arrows) are typed and the composition of morphisms is not necessary to be a morphism. In this mathematical mechanism, the objects of typed category are mathematical abstraction of nodes in Quilian semantic networks, Sowa's concepts, Chen's entities or Allen's events, while typed morphisms are Sowa's conceptual relations, Chen's relationships or Allen's time interval relations. The morphism types in this paper refer to abstractions of different semantics inherent in these links and relationships, such as is-a, part-of and before or after an action. Based on these definitions, this paper devised a way to model knowledge complexity reducible process and the mathematical characterizations of knowledge completion. As the knowledge used in this thesis is all circumscription and default logic based, so the reduction of the complexity of knowledge is not key issue for the VirtualGPS project. However, The functors used in areas of linking problem space to solution space or linking the pieces of isolated knowledge together to get the knowledge completion are very useful for this project.

These previous works have proved that Category Theory can be used as a formal mathematical basis for object-oriented knowledge applications. However, these researchers have focused on specific aspects without providing a unified mechanism for the multi-functional knowledge-based system, and the implementation part of these previous works has lagged behind. Moreover, this project focuses on addressing knowledge interpretation and knowledge processing (e.g. store knowledge) in a direct manner, without using a mechanism to model them, and then use a separate mathematical theory to implement these models. Therefore, based on the aforementioned investigation findings, the researcher in this project devised a categorical mechanism for modelling and implementing the knowledge-based system, which contains three major parts: a categorical object model; a categorical software design process; and an inference identifying square (natural transformation square).

The categorical object model devised in Section 3.4 is used to model structures of entities in knowledge for knowledge acquisition and knowledge representation. Section 3.7 and 3.8 are two examples of using the categorical object model to model structured knowledge for knowledge base design and to model objects for database schema design respectively. The categorical software design process defined in Section 3.5 is used to model the whole system architecture and business logics in the system. The inference identifying square is used to specify how inference properties are interacted with inference rules in detail. The Section 3.6 is an example of using this square to model the comparison process by using the comparison rules defined in GPS.

3.3 Category Technique Rationales

To development of a software system, a suitable system modelling strategy needs to be chose and clarified in advance for the whole system design process. As the VirtualGPS is a knowledge-based system, the system design should focus on the knowledge/application modelling and database modelling. To avoid the error-prone and misunderstanding process of mapping the data stored in a database into objects in the knowledge base of the VirtualGPS system or vice versa, researchers in this project devised a unified modelling mechanism which can be used both on the application side and on the database side. In this research, Category Theory is used for solving six important factors relating to the design of the VirtualGPS:

1. Category Theory is applied to define a stable measurement procedure. As claimed by Kappel and Vieweg, the process modelling step is of vital importance to manufacturing applications (Kappel and Vieweg, 1994 [70]). Within which, measurement procedures are key to the final quality of a manufactured product. Category Theory serves well in terms of improving the stability of a selected measurement procedure.
2. Category Theory was adopted to acquire and represent the knowledge extracted from existing GPS matrixes. Category Theory has rich semantic constructs and notations in both diagrammatic formalisms, as in geometry, along with symbolic notations as in algebra. Diagrammatic constructs were used to handle complexity issues whilst symbolic notations were used for proofs and computation (Nelson and Rossiter, 1994 [65]). It guides knowledge-base designers a tool to build categorical object models that can clearly reflect knowledge-base structures with formal mathematical formulizations. Moreover, the Category Theory can be sufficiently used to unify traditional knowledge representation mechanisms, such as frames and rules, to provide a high degree of unification in knowledge acquisition and representation processes.
3. The system architecture can also be described by Category Theory, with a high level of abstraction. The knowledge bases, mappings, and database schemas with multi-level architectures can be more naturally modeled by the multi-level framework of Category Theory. This can be done by using features such as subcategories, functors, natural transformations, fibration and adjointness in the modularized manner. Thus, the multi-level relationships and constraints will not be lost during the implementation, and it also facilitates the incremental development (data refinement process) for future expansion. Designers are able to add new features or update existing features in the system without requiring major changes on the software structure. The Category Theory can also devise a topological graph to model the deployment of system components on computing resources.
4. The categorical object model was also used in the “categorical” Database

Management System (DBMS) developed for the VirtualGPS system. In this project, system designers used Category Theory to model the software framework and the knowledge base of the system. So it was wise to use same modelling mechanism in the database side since there is no need to program any mapping between the data in the database and the data in the application. Thus, an object-oriented DBMS fully supporting the categorical object model is required in this project. Comparing with a conventional relational data model based on the Set Theory, this categorical DBMS relies on the Category Theory to provide a rigorous mathematical foundation, which can support handling of complex data structures and manipulations. For example, as discussed in section 3.4.3, the identifier for a class category is the vertex representing a "limit" in the universal cone, so each internal relationship/method arrow existing between internal objects must commute with the arrows from the initial object to the corresponding internal objects that are involved in the internal relationship or method.

5. Both dynamic features (e.g. methods) and static features (e.g. attributes, objects) of the object-oriented database schemas can be modeled uniformly using arrows. The type and definition of arrow will determine what its role actually is. This is much better than Set Theory that uses two different notions − set and function to represent static and dynamic aspects in separate way.
6. Category Theory is a form of Constructive Mathematics. All notions, no matter in diagrammatic or symbolic formats, are themselves formal proofs. It formulates complex object structures and behaviors from basic constructs and notations. This ensures a clear structure for object storage and the algebraic manipulations based on categories. Thus, diagram chases and algebra deduces can be used to prove the integrity and consistency of the whole system after any updating, deleting or addition operations.

All in all, Category Theory provided a good unified tool that enabled the system design from high-level system architecture down to the knowledge base, and from static aspects to dynamic aspects in same mathematic mechanism. Thus, different modelling powers from different modelling mechanisms can be unified in single mathematical foundation. Moreover, it provides good abstractions that provide a deep insight into the essence of knowledge and knowledge processing, which can not be obtained simply from a large number of details.

3.4 Categorical Object Model

For this project, application objects are extracted from GPS- matrixes, which can be used to synthetically guide the whole manufacturing life cycle including function, specification, manufacture and verification. These objects have complex structures (especially in a hierarchical level format) and relationships with various types. In this section, a categorical object model will be discussed. This object model is intended to support the core mandatory features for a data model to be qualified as object-oriented data model (object model) claimed in the manifesto for object-oriented DBMSs published by Malcolm Atkinson et al. in 1990 (Atkinson et al., 1990 [35]). Based on the manifesto and the other data model reviewed in section 2.2 of Chapter 2, the core features for the categorical object model are summarized as following:

- Complex class and object support
- Attribute and Method
- Object identity
- Encapsulation
- Types
- Relationships/dependencies
- Inheritance/ class hierarchies
- Integrity/Consistency checking

A brief introduction on key notions of Category Theory used in following sections can be found in section 3.1.

3.4.1 Complex Class and Object Support

The class notion used in the proposed object model is similar to the type notion used in type system. It contains the common features of a set of related objects. In object-oriented applications, real world entities are represented as classes and the instances of entities are represented as objects. From this point of view, the categorical object model uses the "category" notion to represent a class denoted as $\boldsymbol{CLS}_i$ ($1 \leq i \leq n$, n is the number of classes in the database schema). The categorical object model represents all attributes defined in a class as internal objects in a category (internal objects can be another categories or primitives) and each category $\boldsymbol{CLS}_i$ has a collection of arrows mapping between internal objects where these arrows can either represent behaviors (methods) or associations (dependencies). A category with a set of arrows inside in it can be used to describe the structure of a class. All arrow constructs such as composition and dependency must conform to basic laws defined in Section 3.1.1. The instance category $\boldsymbol{OBJ}(j)_{CLS_i}$ ($1 \leq j \leq m$, m is the number of objects created on $\boldsymbol{CLS}_i$) denotes the instance object created on the class $\boldsymbol{CLS}_i$. For example, if $\boldsymbol{OBJ}(1)$ represents an object, $\boldsymbol{CLS}_2$ represents a class category, so $\boldsymbol{OBJ}(1)_{CLS_2}$ indicates that $\boldsymbol{OBJ}(1)$ is the first object created on $\boldsymbol{CLS_2}$. The notation $\boldsymbol{CLS}$ represents all class categories defined in the database schema and $\boldsymbol{OBJ}_{CLS_i}$ represents all instance categories (objects) created on the $\boldsymbol{CLS}_i$. Therefore, this thesis uses the term of class category equaling to class and the term of instance category equaling to object. The both kinds of categories are required to be stored in the categorical database: class categories are stored as metadata and instance categories are stored as real application data. In practice, the creation of an instance object on a class category $\boldsymbol{CLS}_i$ is actually assigning a functor from a class category $\boldsymbol{CLS}_i$ to an instance category $\boldsymbol{OBJ}(j)_{CLS_i}$. Every class modeled in the categorical object model is labeled with a unique meaningful name that is the same as class name defined in object-oriented programming (e.g. Java programming). The name is a special label used to identify classes and to convey the meaning of classes. In this model, a class category is actually a kind of finite complete category.

3.4.2 Attribute and Method

The categorical object model represents all attributes defined in a class as internal objects in a category. $ARR_i = \{ f_j \mid 1 \leq j \leq v$, v is the number of arrows in the category$\}\}$ is used to represent all arrows in a category $\boldsymbol{CLS}_i$. Each category $\boldsymbol{CLS}_i$ has a collection of arrows where these arrows can either represent behaviors (transformations) or associations (dependencies). In this project, behaviors correspond to methods defined in the class, and associations correspond to dependencies between attributes of the class. In this model, the notation *ME* is used to represent a set of method arrows and *DP* is used to represent a set of functional dependency arrows. Each arrow is named uniquely with names of methods or dependencies. As introduced before in Section 3.1.1, every arrow *f* has a domain *dom(f)* and a codomain *cod(f)*. Thus, if ATT_i is used to represent all internal objects (attributes) in the category $\boldsymbol{CLS}_i$, then ATT_i is a collection of $\{dom(f_j) \cup cod(f_j)\}$ for all $\{f_j \mid 1 \le j \le u$, u is the number of arrows in the category$\}\}$. The transitivity of functional dependency arrows must conform to transitivity law defined in Category Theory.

3.4.3 Object Identity

Object identity is a unique key for applications sharing of objects. Assigning a unique identifier to every instance of database entities (classes) is vital important for object-oriented DBMS. The vertex of the universal cone (limit) can be used to model the unique identifier. If viewing the universal cone as a category, the vertex of the universal cone is actually the initial internal object in this category with an arrow from itself to every other internal object in the category. This kind of dependency arrows has an exclusive name − "attribute arrow" in this categorical object model. Therefore, the unique identifier can be represented by ID_i (the initial object of $\boldsymbol{CLS}_i$). The initial internal object stores a unique system automatically generating identifier value. This ID value cannot be modified by applications and is independent of how an object is manipulated or structured. By modelling the database in this way, database users have no need to define keys (primary keys or candidate keys).

3.4.4 Encapsulation

For object-oriented applications, the good encapsulation means both related data part and operation part should be treated as a unit (class) with clearly defined interfaces, so related information can be changed as a whole. The "category" notion of Category Theory can satisfy this encapsulation principle: data part is modelled as a set of internal objects of a category while operation part is modelled as operation (method) arrows between internal objects. Furthermore, a message passing is defined as function arrow mapping from one method arrow to another method arrow. This mapping can occur within a category (intra-class) or between different categories (inter-class). In this case, a higher level category can be formed ─ the arrow category, denoted as $\boldsymbol{C}^{\rightarrow}$ that uses all arrows in the category $\boldsymbol{C}$ as internal objects with function arrows as internal arrows mapping between internal objects in $\boldsymbol{C}^{\rightarrow}$ (Nelson, 1998 [32]; Barr and Wells, 1996 [56]). For example, if there is a message φ_f delivering information (function invocation, signals, and data packets) from an arrow (method m_p) in class $\boldsymbol{CLS}_i$ to an arrow (method m_q) in class $\boldsymbol{CLS}_j$, then φ_f can be represented as $\varphi_f : m_p \rightarrow m_q$ ($m_p \in ME_i$, $m_q \in ME_j$, ME_i and ME_j are methods arrow collections in $\boldsymbol{CLS}_i$ and $\boldsymbol{CLS}_j$) with diagrams in Figure 3.14 commutes as $m_q \circ \varphi_{f_{dom}} = \varphi_{f_{codom}} \circ m_p$.

$$\begin{tikzcd}
\mathrm{dom}(m_p) \arrow[r, "m_p"] \arrow[d, "\varphi_{f_{dom}}"'] & \mathrm{cod}(m_p) \arrow[d, "\varphi_{f_{codom}}"] \\
\mathrm{dom}(m_q) \arrow[r, "m_q"'] & \mathrm{cod}(m_q)
\end{tikzcd}$$

Figure 3.14: Commutative diagram for message passing.

In fact, if $\boldsymbol{CLS}_i$ and $\boldsymbol{CLS}_j$ are different class categories and there is a functor mapping from ***CLS*** to $\boldsymbol{CLS}^{\rightarrow}$, then the message passing process can be regarded as a natural transformation mapping between pairs of ***CLS*** to $\boldsymbol{CLS}^{\rightarrow}$ functors. In addition, the types of message senders and recipients conform to the types of method arrows participating in the message passing.

3.4.5 Types

In Category Theory, one discrete item is identified by the single category ***1*** (Nelson et al., 1994 [66]). Hence, typing can be added to show the types upon which the item is taken from in form of $\boldsymbol{1_{TYP}}$, where ***TYP*** can be the base types in object-oriented programming language (e.g. String), other class categories, or other defined complexity such as arrows, arrays and lists. When ***1*** denotes class categories or arrows, the values of ***1*** are names of these class categories or arrows. Arrows are typed in form of f: $a \xrightarrow{1_{TYP}} b$, where the a is the source internal object, b is the target internal object, $\boldsymbol{1_{TYP}}$ is the type. In arrow composition situation, such as the f: a $\xrightarrow{1_{TYP1}}$ b and g: b $\xrightarrow{1_{TYP2}}$ c, the $f \circ g$: a $\xrightarrow{1_{TYP1\times TYP2}}$ c, where the $\boldsymbol{1_{TYP1 \times TYP2}}$ is the type composition.

3.4.6 Relationships

The generalization abstraction between class categories are modeled using "forgetful" functor mapping from subclass to superclass. Forgetful functor is a structure preserving mapping from one category to another category with some attributes and methods dismissed. The aggregation abstraction between class categories are modeled using "faithful" functors which inject one category into another category while preserving its structure. For other common relationships occurred at category level, comparing with relational algebra that defines relationship as projection or cartesian product, the categorical representation of relationships is product. At category level, the product is formed by categories and functors instead of internal objects and arrows defined in section 3.1.2. See Figure 3.15.

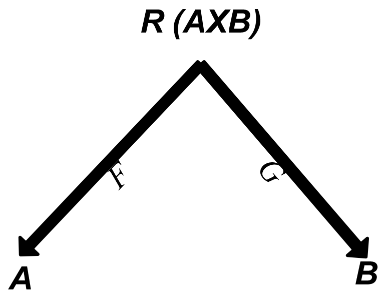


Figure 3.15: The relationship *R* between *A* and *B*.

Normally, the *F* and *G* are forgetful functors (mapping some of related arrows into the target category). The product relationships in this object model are also represented as categories $\{ \boldsymbol{REL}_i \mid 1 \le i \le w$, *w* is the number of product relationships in the database schema$\}$. Therefore, in Figure 3.15, ***A*** and ***B*** are class categories, ***R*** is a relationship category between them. The relationship category ***R*** is represented in the form $\{< ID_i^R, r' > \mid r' \in \wp$(arrow set in ***A*** and arrow set in ***B***)$\}$, where ID_i^R is the identifier of the relationship category that is assigned by DBMS automatically, and *r'* is any information generated from this link. The link itself contains an element in the powerset of arrow set in ***A*** and arrow set in ***B***. The related arrow compositions in ***A*** or ***B*** will be preserved. For every *r* in ***R***, $F(r) = a$ in instance set of *A* and $G(r) = b$ in instance set of ***B*** must exist for referential integrity.

The functors shown in the Figure 3.15 can be typed into universal monomorphisms (***M***), universal epimorphisms (***EP***), and universal isomorphisms (an arrow that is both monomorphism and epimorphsim is called isomorphism, ***ISO***). Therefore, functors *F* and *G* can carry useful information (constraints) relating to relationships:

- If *F* is in type of ***M***, then each instance of ***A*** is involved only once in the instance set of ***R***. However, if *F* is not ***M***, then an instance of ***A*** may be involved more than once in the relationship. There may have some instance of ***A*** which does not participate in the instance of ***R***, so the membership of ***A*** is optional. The same situation applies for functor *G*.
- If *F* is in type of ***EP***, then every instance of ***A*** is involved at least once in the

instance set of ***R***. Therefore, all instances of ***A*** participate in the relationship, which means the membership of ***A*** is mandatory. However, if *F* is not ***EP***, then not every instance of ***A*** is involved in the relationship and the membership of ***A*** is optional. The same situation applies for functor *G*.

- If *F* is in type of ***ISO***, then every ***A*** must participate once and only once in the relationship and the membership of ***A*** is mandatory. The same situation applies for functor *G*.

In the categorical object model, membership is represented by typing of the functors, which is much formal than the traditional way using labels. This modelling method can be extended to satisfy the modelling of n-ary relationships by using n-ary product construction. Multiple relationships between same class categories are identified by different relationship categories. For example, $\boldsymbol{R_1}(\boldsymbol{A}\times\boldsymbol{B})$ and $\boldsymbol{R_2}(\boldsymbol{A}\times\boldsymbol{B})$ are two different relationship categories between class categories ***A*** and ***B***. To help designers to comprehend class category and product relationship between class categories, the following two rules are devised for calculating the cohesion for a class category as well as the coupling for a product relationship category:

- The cohesion of a class category (*Ce*) can be calculated as the average number of the dependencies and behaviors for each internal object. Let *Rn* be the number of dependency and behavior arrows between internal objects in a class category, which should also include all dependency and behavior arrows of the sub-categories (except all attribute arrows). Let *On* be the number of internal objects in the class category. Then, $Ce = Rn/On$.
- The coupling of a product relationship (*Cp*) can be calculated by the using number of internal objects in a relationship category, which are gathered from all class categories that participate in this relationship link. Let *In* be the number of internal objects in a relationship category, which are gathered from participating class categories and *En* be the number of the rest internal objects in these participating class categories. Then, $Cp = In / (In + En)$, where *Cp* range from 0 to 1.

The above two rules illustrate the fuzzy logic applied in the categorical object model at different hierarchical levels:

1) Attribute values: the [0, 1] interval can be used to express the explicit uncertainty that affects an attribute value.
2) Class extents: a class category can be extended in a fuzzy way to define its

domain in the interval [0, 1].

3) Relationship coupling: use appropriate truth scales for expressing strength or connection uncertainty.

3.4.7 Inheritance/Class Hierarchies

The "pushout" construction is an extension of the coproduct construction which provides the complex sum structures (i.e. amalgamated sums) in the categorical object model for two or more class categories. This is superior to the simple disjoint unions (Nelson, 1998 [32]). The inheritance hierarchies in this model can be naturally constructed by coproduct construct since the ancestry of each class in the hierarchy is preserved through using the pushout structures. Let $\boldsymbol{CLS}_3$ be a class category representing a subclass category of class category $\boldsymbol{CLS}_1$, the $\boldsymbol{CLS}_3$ contains a set of arrows ARR_3 (methods or dependencies) and internal objects ATT_3 (attributes). The coproduct $\boldsymbol{CLS}_1 + \boldsymbol{CLS}_3$ is the disjoint union of the arrows ($ARR_1 + ARR_3$) and the attributes ($ATT_1 + ATT_3$). Figure 3.16 shows an example that the class category $\boldsymbol{CLS}_3$ inherits from the class category $\boldsymbol{CLS}_1$.

$$\begin{tikzcd}
& CLS_1 + CLS_3 & \\
CLS_1 \arrow[ur, "\pi_i"] & & CLS_3 \arrow[ul, "\pi_j"'] \arrow[ll, "inf"]
\end{tikzcd}$$

Figure 3.16: Coproduct diagram for class inheritance.

In Figure 3.16, $\boldsymbol{CLS}_1$ contains all attributes (internal objects) and methods (arrows) for a parent class category and $\boldsymbol{CLS}_3$ contains attributes (internal objects) and methods (arrows) for a subclass category. The $\boldsymbol{CLS}_1 + \boldsymbol{CLS}_3$ is the disjoint union of attributes (internal objects) and methods (arrows) of $\boldsymbol{CLS}_1$ and $\boldsymbol{CLS}_3$ combined together. The arrow *inf* shows the direction of the inheritance.

3.4.8 Implementing Operations

In order to improve reusability, communication and class sharing, interfaces and abstract methods are often used in real world applications, especially some large applications. The interface is like a skeleton, which contains only method signatures and variables. Methods must be public, abstract and their variables must be public static final. It is advisable to design relatively large applications using interfaces because it makes the whole system easier to modify, extend and integrate new features. To start with, system designers may only have one implementation of a given interface, but when slightly different behaviours are required in special circumstances during the design process, designers only need to devise a class that conforms to one of the existing interfaces and it will drop in place without major modifications. Interfaces also allow programmers to adopt a class from a different hierarchy to work in an existing application. The class only needs to declare itself implementing a specific interface, provide the necessary methods and it can be integrated directly as if it were created for the job. In the categorical object model, the index category, category and functor are used to model the interface concept. Interface is modelled as an index category of a concrete class category, which contains only a collection of internal objects typed in "$\boldsymbol{1}_{\boldsymbol{final\_static}}$" and a set of methods typed in "$\boldsymbol{1}_{\boldsymbol{Abstract}}$". A functor in type of "$\boldsymbol{1}_{\boldsymbol{Implements}}$" is used to map from the index category $\boldsymbol{T}$ to the class category $\boldsymbol{C}$ while preserving the structure of index category $\boldsymbol{T}$ in $\boldsymbol{C}$. See Figure 3.17.

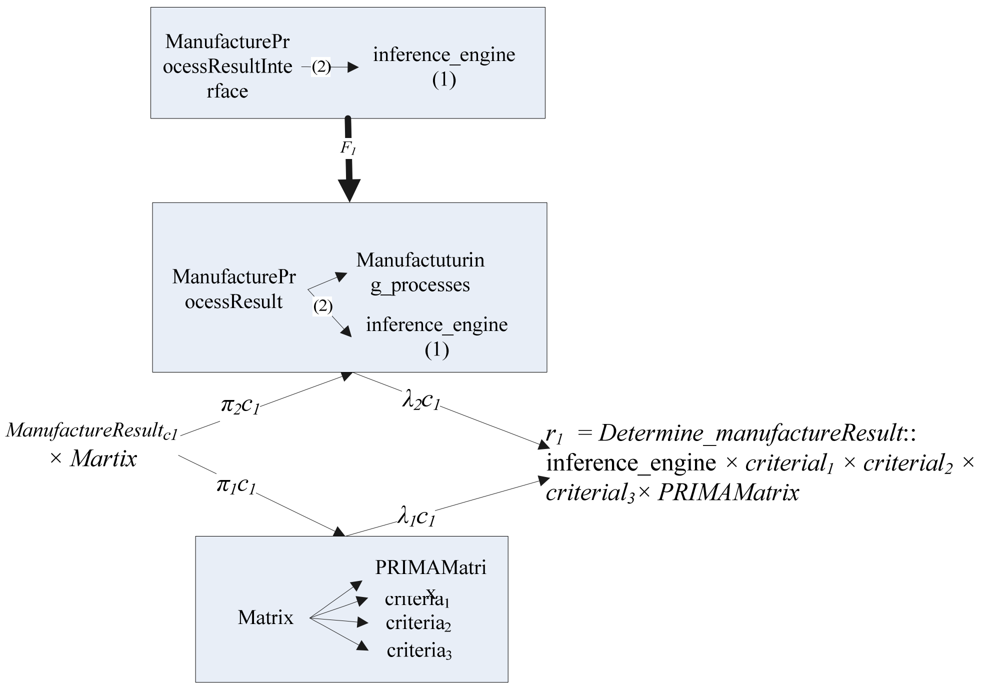

ManufactureProcessResultInterface
(2)
inference_engine (1)
F_1
ManufactureProcessResult
Manufactuturing_processes
(2)
inference_engine (1)
$\pi_2 c_1$
$\lambda_2 c_1$
$ManufactureResult_{c1} \times Martix$
r_1 = Determine_manufactureResult:: inference_engine × $criterial_1$ × $criterial_2$ × $criterial_3$ × PRIMAMatrix
$\pi_1 c_1$
$\lambda_1 c_1$
Matrix
PRIMAMatrix
$criteria_1$
$criteria_2$
$criteria_3$

Figure 3.17: A part of categorical object model for determining the manufacturing processes.

Figure 3.17 shows an implement functor F_1 mapping from the index category "*ManufactureProcessResultInterface*" to the concrete class category "*ManufactureProcessResult*". The index category "*ManufactureResultProcessInterface*" (interface) contains an indexed arrow *(2)* and an indexed method "*inference_engine*" *(1)*.

3.4.9 Physical Storage Linkages

In the physical storage level, both class categories and their instance categories will be stored in the files, where class categories store metadata of class and instance categories store real values. Therefore, there need to define linkages linking metadata in class categories to metadata in internal objects (attributes/lower level categories) and class categories to their corresponding instance categories. Functors are used to record these linkages.

3.4.10 Integrity/Consistency Checking

Integrity or consistency checking in the categorical object model contains two levels: inner category level and inter category level. Inner category level integrity refers to ensure that every internal object should be typed and 3NF should be enforced to eliminate partial and transitive functional dependencies on IDs. To satisfy this, categorical object models used in DBMS should remove all functional dependency arrows between internal objects (except those attribute arrows) in those class categories defined for modelling of the knowledge bases. The inter category level integrity is the referential integrity that ensures a category (class, instance, relationship) actually exist when they are referred by other categories. Therefore, by using diagram chasing, when updating or deleting categories that reside on the target side of arrows or functors, the source side of these arrows or functors must do the corresponding deletions or updating.

3.4.11 Query

To provide manipulation capability for the categorical object model, an object query language is also produced based on functor mappings and functor compositions. In this query strategy, the inputs and outputs of queries are all instance categories associated with either certain class or relationship categories. The forgetful functor is used to choose some of necessary arrows of a class category as the "*Select*" clause in SQL language did. The detail example for the query strategy is illustrated in Section 4.4.1.5 of Chapter 4.

3.4.12 Statement

According to the definition, a data model contains logical concepts and mechanisms to describe how data is represented and accessed. Therefore, the fundamental paradigms for choosing a suitable data model are (Kappel and Vieweg, 1994 [70]):

- Whether the information of the application can be easily mapped to the data model.
- Whether the data model is powerful and clear enough to representing complex data structures.
- Whether the data model is easily to be implemented in the programming.

The above eleven points 3.4.1-3.4.11 have justified that the categorical object model based on Category Theory is a suitable data model for this project.

3.5 The Categorical Design Process

Because of the large size and high complexity of the VirtualGPS knowledge-based system, researchers in this project needed to provide a unified theoretical framework for representing the system through appropriate mathematical formalizations. The aims for defining the unified theoretical modelling strategy for describing the VirtualGPS are:

- Reducing the complexity of managing the whole system through clearly and gracefully representing modules and their interconnections. The modelling strategy should also facilitate the realization of new modules and extension of the software system. Moreover, such strategy should be formal and avoid high level ambiguities.
- Close the gap between software designs and implementations. The implementation aspect of this system contains a set of modules which in turn contain a set of components. Therefore, clear definition of the business logics among different modules or components are of vital importance in the design stage of the system development.
- Providing rich set of semantic means for easing implementation. By offering the sufficient semantic constructs with a high level of abstraction, designers can concentrate on describing the semantic aspects of applications rather than representational issues.

These aims indicate Category Theory is an excellent tool. The overall VirtualGPS system architecture contains two major parts: system modularized framework and system deployment graphs. The system modularized framework focuses on specifying the functions of all the modules, their mutual interactions and transformations. The system deployment graph is emphasized on specifying the system allocation and how system users can access this system. Therefore, the basic categorical principles that researchers adopt in dealing with these two parts are:

1. As VirtualGPS contains a set of modules which in turn contain a set of components, modules are corresponding to categories whose internal objects are corresponding to components. The detail representation of components can be modeled as lower level categories and functors are used to connect lower level categories with their higher level categories.
2. The dependencies such as constraints or interactions between different modules or components are modeled by using the product constructs of Category Theory. The components, modules and dependencies between them can be realized by using the two rules discussed in Section 3.4.6. For example, if the coupling of a product relationship (*Cp*) between two components are very high (near to 1), so either a module or a new bigger component should be organized for holding these two components.
3. The extraction of similarities between different components is modeled by using the coproduct construct of Category Theory. This can also be useful in realizing new components or modules. For example, in order to improve the reusability and independent ability of the system, some fragment programming codes shared by several components will be removed from the disjoint union of the corresponding components to form a separate new module.
4. The coordination protocols (e.g. message/signal passing, invocation mechanism and communication rules) and business logics/rules (e.g. information exchange or communication) among modules or components are modeled by using the structure preserving construct − natural transformation.
5. The deployment topologic graphs of the VirtualGPS system are represented by using the diagram notion of Category Theory.

Based on the above five points, a unified refinement design process can be developed using Category Theory. In traditional software designs, the Unified Software Development Process (USDP) is used throughout the whole lifecycle. The USDP is a software development process, which includes a set of activities needed to transform a user's requirements into a software system. It is a generic process framework that can be specialized for a very large class of software systems, for different application areas, different types of organizations, different competence levels and different project sizes. The USDP is component-based design process, which uses the Unified Modelling Language (UML) when preparing all blueprints of the software system

(Jacobson, 2004 [71]). The basic design steps by using UML in a USDP incremental development process are:

1. The first step is the business map design. This step is used for capturing the requirements of users. The outputs of this step are business maps and use cases. A business map shows the business scope of the target software system. Based on the business map, a set of use cases can be produced to detail specify the meaningful interactions within this computerized system. The use cases often become modules in the final software system.
2. The Second step of USDP process is to create the analysis model from the use cases model. The analysis model is used to obtain a more precise specification of requirements than the requirements captured during use case modelling. The output of this step is a set of the initial analysis classes for the software system. The analysis classes often become components in the final software system.
3. The third step is to create the design model from the analysis model. In this step, the design classes in the design model are defined to trace the analysis classes in the analysis model. The design classes are refined from the analysis classes. Therefore, design classes are more adapted to the implementation environment. Several design classes can be organized together to form a reusable component.
4. The fourth step is to create the sequence diagram for realization of every design class in the design model. The sequence diagram shows how the focus-starting at the upper left corner-moves from design class to design class as the use case is performed and messages are sent between design classes.
5. The final step is to divide the design classes in the design model into subsystems based on outputs of points 3 and 4. This step is contributing to form the topology of the system allocations (deployment model).

Points 1 to 3 are actually a refinement and incremental process to determine the real design classes (modules or components) for a software system from initial use cases offered by users. After the components of a software system have been determined, point 4 is the key step to define the potential inter-relationships or inter-activities between modules or components, and help to define interfaces for them. Therefore, the key features of USDP are use-case driven, architecture-centric, iterative and incremental (Jacobson, 2004 [71]). The categorical design process devised in this thesis is also an incremental and refinement process. However, unlike to the USDP from points 1 to 3, the designers need to build different models or diagrams to achieve this refinement process, the categorical design process start by determining some high-level general categories and then to elaborate and refine them by:

- In the horizontal level, general categories may need to be separated into several elaborated small categories. The separated small categories are required to be linked together by using the product relationship construct.
- In the hierarchical level, the internal objects of some higher level categories need to be detailed in the form of lower level categories. The lower level categories are injected into their corresponding higher level categories by using the "faithful" functor.

This refinement process is used to convert a simplified abstract model into a complex implementable model. In order to ensure the consistency between two models, the functor mappings used in this refinement process need to satisfy the following requirements:

(1) **Integrity**: every abstract category and abstract relationship/constraint in an abstract model must map correspondly in its implementable model.

(2) **Composition**: All the transitive relationships in an abstract model should be preserved in its implementable model.

(3) **Completeness**: every target concrete category for a refined relationship in an implementable model is a target abstract category for its corresponding abstract relationship in its corresponding abstract model.

(4) **Pattern reservation**: every concrete category and relationship in the implementable model is part of a pattern with the same structure as in the abstract model.

As it has been proved by Colomb, et al., the functor with fibration feature can satisfy these four requirements above (Colomb et al., 2001 [68]). A functor F mapping from an abstract model to an implementable model with fibration can be defined in detail as:

- Associates to each object $X \in \boldsymbol{A}$ (abstract model) with an object $F(X) \in \boldsymbol{I}$ (implementable model);
- Associates to each morphism u: $\mathrm{X} \rightarrow Y \in \boldsymbol{A}$ with a morphism $F(u)$: $F(X) \rightarrow F(Y) \in \boldsymbol{I}$;
- $F(id_X) = id_{F(X)}$ for every object $X \in \boldsymbol{A}$;

- $F(v \circ u) = F(v) \circ F(u)$ for all morphisms u: X → Y and v: $Y \rightarrow Z$;
- A functor F is a fibration, if and only if for every object $F(Y)$ of $\boldsymbol{I}$ and every map u: $X \rightarrow Y$ *in* $\boldsymbol{A}$, there exists a Cartesian morphism $F(u)$: $F(X) \rightarrow F(Y)$ in $\boldsymbol{I}$. The Cartesian morphism is used to ensure completeness in an implementable model.

This refinement process can also be used to refine the categorical object modelling diagrams defined in this thesis for modelling of structured knowledge refinements. The examples for demonstrating a complete categorical design process for designing the VirtualGPS can be referred in Chapter 5. This section focuses on giving detailed explanations on building a categorical sequence diagram and a categorical system deployment diagram (see subsection 3.5.1 and 3.5.2 respectively).

3.5.1 An Example for Building the Categorical Sequence Diagram

The detailed description on the design of the VirtualGPS system is given in the Chapter 5 of this thesis. This section concentrates on giving a detailed explanation for building the categorical sequence diagram using the example of the comparison process in the Verification module as illustrated in Figure 3.18.

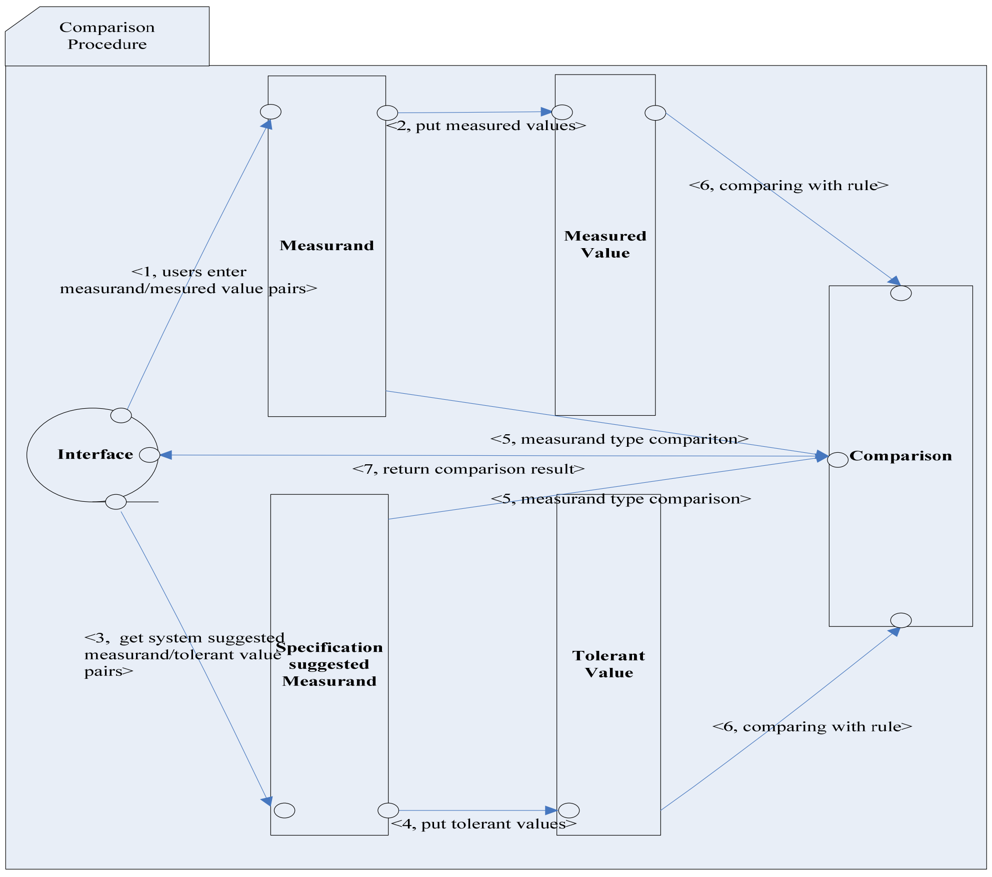

Comparison Procedure
Measurand
Measured Value
Comparison
Interface
Specification suggested Measurand
Tolerant Value
<1, users enter measurand/mesured value pairs>
<2, put measured values>
<3, get system suggested measurand/tolerant value pairs>
<4, put tolerant values>
<5, measurand type compariton>
<5, measurand type comparison>
<6, comparing with rule>
<6, comparing with rule>
<7, return comparison result>

Figure 3.18: The categorical sequence diagram for comparison processes.

The diagram "*ComparisonProcess*", shown in Figure 3.18, is an indexed category with initial internal object "*Interface*" and four other internal objects: "*Measurand*", "*MeasuredValue*", "*ToleranceValue*", and "*Comparison*". These four internal objects are low level categories. Arrows in the categorical sequence diagram are in the type of message arrow. Message arrows are responsible for sending messages between internal objects with two default sending properties in form of "<sequenceNo, messageName>" where "sequenceNo" is used to identify message sending sequence and messageName is a string to describe this message in general.

3.5.2 An Example for Building the Categorical System Deployment Model

The final stage in the categorical design process is to build the categorical system deployment models. For example, based on the sequence diagram of Figure 3.18, the following steps should be adopted to build a categorical system deployment model:

1. Refine the internal objects of the categorical sequence diagram to get the refined low level categories. In Figure 3.18, the initial object "*Interface*" becomes two refined lower level categories "*ParameterReceiver*" and "*ParameterCreator*" that are responsible for receiving and creating measurand/value pairs. In Figure 3.18, the internal objects "*Measurand*" and "*SuggestedMeasurand*" are holding data in same structure, so these two internal objects can be merged to form the class category "*Measurand*". For the same reason, the "*measuredValue*" and "*ToleranceValue*" become the class category "*Value*". Different functor instances are used to distinguish the pairs for suggested measurands with tolerance values from pairs for measurands with measured values entered by users.
2. Link functional related categories to form the components of a software system. In this case, Figure 3.18 clearly shows the comparison process can be separated into three components: interface component, natural transformation square for comparison process, and comparison component. The detailed discussion of the natural transformation square for the comparison process (see Figure 3.22) is demonstrated in section 3.6. The interface component contains the "*ParameterReceiver*" and "*ParameterCreator*" categories. The comparison component contains the class category "*Comparison*" and the class category "*ComparisonManager*" that is responsible for storing the final comparison results.

3. Link functional related components to form modules or subsystems of a software system. A module (subsystem) can consist of components, interfaces, and other modules (recursively).
4. Components and modules are allocated on a system deployment topological graph to form the deployment model. In the same way as other topological graphs that are formed by a set of nodes and edges between nodes, the nodes in the categorical system deployment topologic graph are computational resources such as servers, clients, processors or similar hardware devices while the edges represent relationships or communications between nodes such as Internet, intranet, bus and so on. Figure 3.19 is a categorical deployment topological graph for the comparison process.

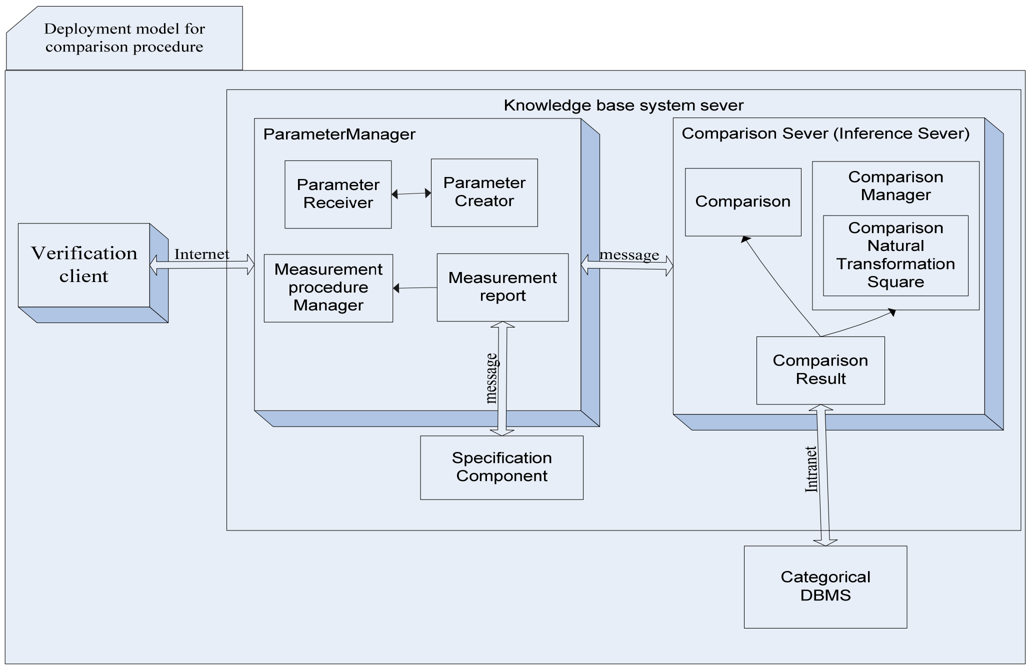


Figure 3.19: Categorical representation of a deployment topological graph.

Figure 3.19 is a deployment topological graph for deploying the software components relating to the comparison process on the physical computational nodes, which is formed in a categorical view: every node is a category and edges between nodes become functors to represent communications. The functors in the deployment models can have loops (i.e. functors from a category to itself just like identity arrow notation) and multi-functors (i.e. functors that have the same source category and the same target category). In Figure 3.19, there contain three categories: "*SurfaceTextureClient*", "*SurfaceTextureModuleServer*", and "*Categorical DBMS*", as well as two typed functors: "*Internet*" and "*Intranet*". Any functor *F* here should mapping from source node to target node while preserving their structures through that: (1) for every component *A* in source catgory ***S***, a component *F*(*A*) should be in the categoty node ***T***; (2) for every relationship or communication *f*: *A*→*B* in ***S***, a relationship or communication *F*(*f*):*F*(*A*) → *F*(*B*) should be in ***T***; (3) every relationship or communication composition in ***S*** should be preserved in ***T***. According to Figure 3.19, the VirtualGPS system should provide an interface for users to enter measured values on the client side. The measured values with measurands will be sent to "*ParameterReceiver*" category which in turn communicates with "*ParameterCreator*" to create measurand/measured value pairs and suggested measurand/tolerance value pairs. Then, these pairs will be sent to the "*ComparisonManager*" that is responsible for holding pairs and creating natural transformation squares. The "*ComparisonManage*r" has a product relationship with "*Comparison*" to form a relationship category "ComparisonResult". The "*ComparisonResult*" is used to store comparison results based on the comparison information in "*Comparison*" and natural transformation squares in "*ComparisonManager*". The detailed information for the construction of these natural transformation squares can be referred in Figure 3.22. Finally, the comparison results will be stored in the categorical DBMS through Intranet (sockets).

3.6 Categorical Representation for the Measurement Theory

One of the main attractions of Category Theory in this project is that it provides a rigorous mathematical foundation to define the measurement theory. As has been successfully proven in the past, the representational measurement theory can be used to define the stability of the measurement procedure (Scott, 2004 [72]; Scott, 2006 [73]). The measurement procedures relating to this project contains three key points in terms of the applied representational measurement theory:

(1) An empirical relational system (ERS); consisting of a set of objects on which a measurand is defined together with the relations between other relevant measurands.
(2) A numerical relational system (NRS); comprising numbers (derived values) and the relationships between them.
(3) A set of mappings; referred as the measurement procedures, map from ERS to NRS, in such a way that the relationships between measurands are matched by relationships between numbers.

A measuring procedure is regarded as mathematically stable, when a “small” difference in the derived values can imply a “small” difference in the measurand. Relationships between measurement values should reflect functional significant properties between the measurands; if not, the measurement is rendered unusable (Scott, 2006 [73]). Since in Topology an open set can be used to define “small” differences between points, the stability condition of measuring procedure can be described using the Topology and Set Theory. In 2004, Scott devised stability corollaries that can be used to justify when a measurement procedure is stable or not using following rules (Scott, 2006 [73]):

> ***Corollary 1***: “*Finite sets of measurands and derived values with partial pre-orders and increasing mappings map one-to-one onto finite topologies with continuous mappings.*”
>
> ***Stability Corollary***: “*If for a measurement procedure, the relational structures of the measurand and the derived values are both partial pre-orders and the mapping between them are also increasing mappings then the measurement procedure is stable.*”

Based on the above rules, if define topologies on the space of measurands and the space of derived values, the stability condition is just a continuous mapping from the measurands to the derived values (if the inverse image of every open set on the topological space of the derived values is an open set on the topological space of the measurands, this is a topological definition of a continuous mapping). Researchers in this project found Category Theory can provide a visual framework to vigorously represent the *corollary1* and *stability corollary* using notions and constructions defined in Category Theory. The following points give a short explanation on how Category Theory represents the stability corollary:

(1) In order to satisfy the stability corollary, both ERS and NRS for a measurement procedure should be partial pre-orders with properties of reflexive and transitive, so categories are used to represent ERS and NRS while arrows inside the category are used to represent partial pre-order. Moreover, objects in ERS or derived values in NRS are represented as internal objects of category. See Figure 3.20.

$$\begin{tikzcd} & C \arrow[loop right, "id_C"] \\ B \arrow[ur, "g"] \arrow[loop left, "id_B"] & \\ & A \arrow[ul, "f"] \arrow[uu, "g \circ f"'] \arrow[loop right, "id_A"] \end{tikzcd}$$

Figure 3.20: Categorical representation of ERS.

In Figure 3.20, the id_A, id_B, and id_C are identity arrows (an identity arrow id_A: $A \to A$, for each object A satisfying the identity law as for any arrow f: $A \to B$, $id_B \circ f = f$ and $f \circ id_A = f$), which are used to satisfy reflexive property of partial pre-order. For the transitive property, if arrow f: $A \to B$ and g: $B \to C$ represent binary relations in ERS, so $g \circ f$: $A \to C$ is the categorical representation of transitive property in partial pre-order.

(2) A high-level notion that is a special type of structure preserving mapping (arrow) between categories named as "functor". The formal definition of a functor can be found in Section 3.1.1 of Chapter 3. From the definition, functor must preserve identity arrow and the compositions of arrows inside categories. Therefore, a functor can gracefully represent increasing mappings between ERSs or NRSs with partial pre-orders defined.

Based on the above two definitions, the categorical way of defining a stability corollary can be restated as:

"If for a measurement procedure, the relational structures of the measurands and the derived values are both partial pre-order categories and the mapping between them is functor then the measurement procedure is stable".

Table 3.1 gives a summary to show the relation between categorical terms and the concepts of representational measurement theory.

Category Theory	**Explanations**	**Representational measurement theory**
Category	Collection of internal objects and arrows	Relational System
Functor	Structure preserving mapping between categories	Structure preserving mapping between relational systems
Natural Transformation	Structure preserving mapping between functors	Comparison

Table 3.1: Categorical terms for representational measurement theory concept.

The development of the stability corollary in a categorical way is beneficial for retrieving useful features from the observable data relating to this project, and ensuring consistency of the knowledge acquisition for this knowledge-based system. Moreover, by adding the “natural transformation” notion of Category Theory, the whole verification procedure can be refined as Figure 3.21.

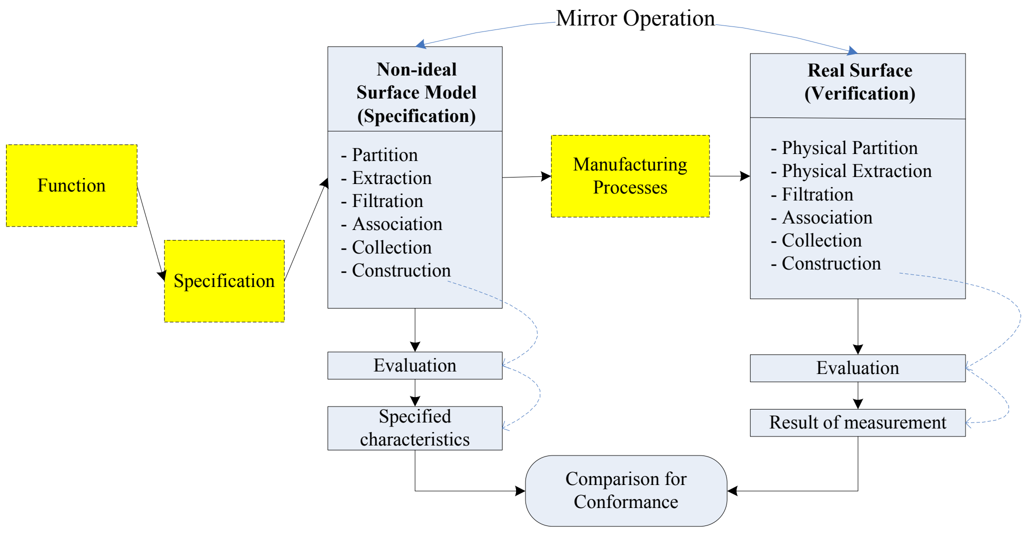


Figure 3.21: Comparison between specification and verification.

The Figure 3.21 also shows a refined general GPS model for the VirtualGPS system. To ensure the stability of measurement, relational structures of measurands in ERS and derived values in NRS of a measurement procedure must be partial pre-order categories.

In verification module of the VirtualGPS system, the inference rules are comparison rules, the inferred properties are measurand/value pairs and the inference results are {*accept* or *not accept*}. There are two kinds of measurand/value pairs: the suggested GPS parameter/tolerance value from Specification component of the VirtualGPS system and the measurand/measured value inputted by users. The mappings in each pair should be defined as functors. Therefore, every measurement procedure must have functors mapping from measurands to the measured values while preserving the internal partial pre-order structures. As a natural transformation provides a feasible way for transforming between functors while respecting the internal structure of the categories involved, the final comparisons are achieved by natural transformations with comparison rules. Figure 3.22 shows an example of the comparison process in the categorical view (inference identifying square).

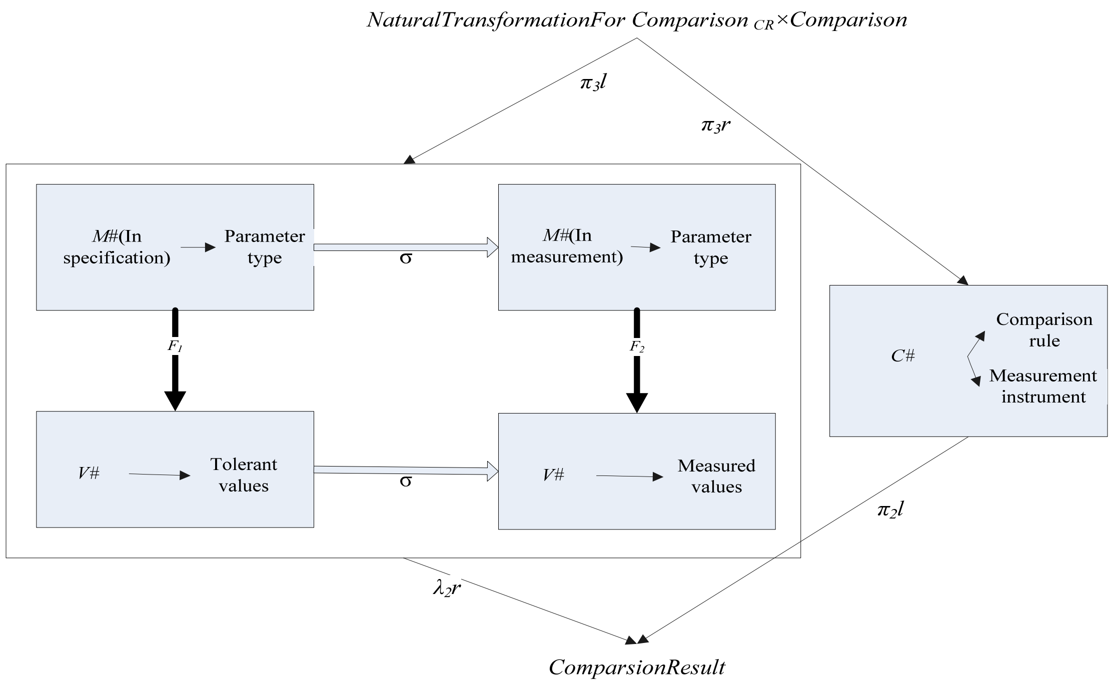


Figure 3.22: Categorical view of comparison processes.

In Figure 3.22, F_1 and F_2 are functors mapping from partial pre-order category "*Measurand*" to partial pre-order category "*Value*". The σ is natural transformation mapping from F_1 to F_2. The F_1, F_2 and σ form a natural transformation square. Figure 3.22 also shows a 2-ary pullback relationship structure between a natural transformation square and a class category "*Comparison*".

This example also shows how to model the interrelationships between inference properties, inference rules and inference results:

(1) Functors are used to link inference properties together.

(2) Natural transformations are used to get the solution space from the problem space with respect to inference rules while preserving the structures of inference properties.

In the some simple situations that we do not need to keep the linkage structures between inference properties, so functors can be directly used to map from problem space (inference properties) to solution space (results). Moreover, the reasoning power of Category Theory can be used for the knowledge deduction for the VirtualGPS system by using the equalizer and co-equalizer constructs (Lu, 2005 [69]). For example, the Manufacture component of the VirtualGPS can be used to determine the manufacturing processes: to select suitable manufacturing processes to match the specification of the designed product. The Figure 3.23 shows an example of coequalizer diagram for reasoning the suitable manufacturing processes.

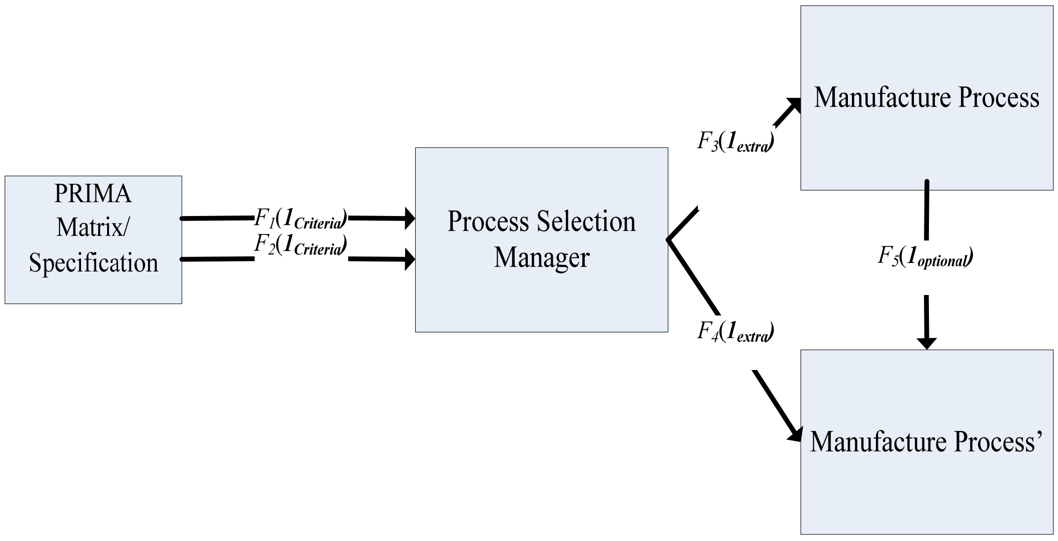


Figure 3.23: The coequalizer for manufacturing process reasoning.

In Figure 3.23, the F_1 and F_2 represent a set of criteria applied to the inference properties (e.g. material and quantity of PRIMA Matrix or limit value, texture lay and cut-off wave length gathered from Specification report). If any inference property a with F_1 $(a) \neq F_2$ (a) (the same property under different inference criteria may get different manufacturing process suggestions), the coequalizer functor in the coequalizer diagram can equalize in the way that for all these a, F_3 $(F_1$ (a)) = F_3 $(F_2$ (a)). This means that the different manufacturing processes suggested by different criteria for same inference property will be unified with extra considerations such as economic considerations, and typical applications. If this final unification has multiple results, F_5 is used to link them together with weight value calculations. In real applications of this case, the coequalizer can be extended to multiple dimensions. In order to calculate weight of different manufacturing process suggestions, fuzzy logic is applied in this project. There has a knowledge representation problem for traditional knowledge-based systems to handle uncertain or incomplete information. To represent vagueness or uncertainty, fuzzy logic is developed, with a continuous range of possibilities from 0.0 to 1.0 for uncertainty (an example can be found in Section 5.4.3) (Sowa, 2000 [74]).

3.7 Categorical Representation for the Knowledge Base

As pointed out by Lu, the development of a mathematical tool to deal with structural properties of knowledge is a basic part of knowledge science (Lu, 2005 [69]), this project provides an innovative way for manufacturing engineers to establish knowledge bases derived from GPS raw standards without requiring specialised expert computer skills. The knowledge base of the pilot system contains four derived sub-knowledge bases (modules): Surface Texture, Form, Size and Position and each module contains four derived sub-components: Function, Specification, Manufacture and Verification. The detail introduction on the architecture of the VirtualGPS system can be referred in Section 5.1 of Chapter 5. The following two subsections explain the modelling of the knowledge in Function and Specification components for the Surface Texture module in Category Theory terms. The other two – Manufacture and Verification – were modelled in similar manner.

3.7.1 Knowledge Modelling in Function Component for Surface Texture

In the Function component, Category Theory is applied through representing function requirements in a so-called "pattern language", which guides the inference engine to generate a function performance report highlighting the suggested specific surface roughness parameters according to the inputted function performance requirements. A single pattern in a pattern language is defined as a common problem or decision with its best solution in a target task. Each pattern has a name, a descriptive entry and cross-references to other patterns. A pattern language is made of several linked patterns that should be organized in a logical and semantic structure as a spoken language in a specific problem domain. The pattern language is used here for facilitating function decomposition and to structure the connection process. This section gives an example on using the partial order set and the product order of Category Theory to represent and record decomposition alternatives (Neggess and Kim, 1998 [75]).

A partial order is a binary relation R over a set $\boldsymbol{S}$, which is reflexive, antisymmetric and transitivity. The set $\boldsymbol{S}$ with a partial order is called a partially ordered set (poset). The function performance report generated from the Function component contains six patterns specified in the explained pattern language format. These patterns are connected with each other by the context of each pattern, and ordered by the design sequence:

- **Pattern 1** specifies the surface requirements.
- **Pattern 2** analyses the functional performances according to the output of Pattern 1.

- **Pattern 3** selects suitable specification to ensure the surface functions correctly.
- **Pattern 4** suggests a function correlation approach between surface texture parameters and the functional performances.
- **Pattern 5** provides an alternative route through the surface change monitoring approach to find the relations between functional performances and surface parameters.
- **Pattern 6** specifies the tolerance values for the parameter selected from pattern 4 and 5.

In this project, the pattern language provides some possible solutions allowing users to make their own judgements. Every pattern in the Function component is represented as a class category which contains seven internal objects: name, context, problem, solution, forces, examples, next pattern. All of them are represented in posets. As illustrated in Figure 3.24, all patterns are connected with each other, which form also an integrated poset.

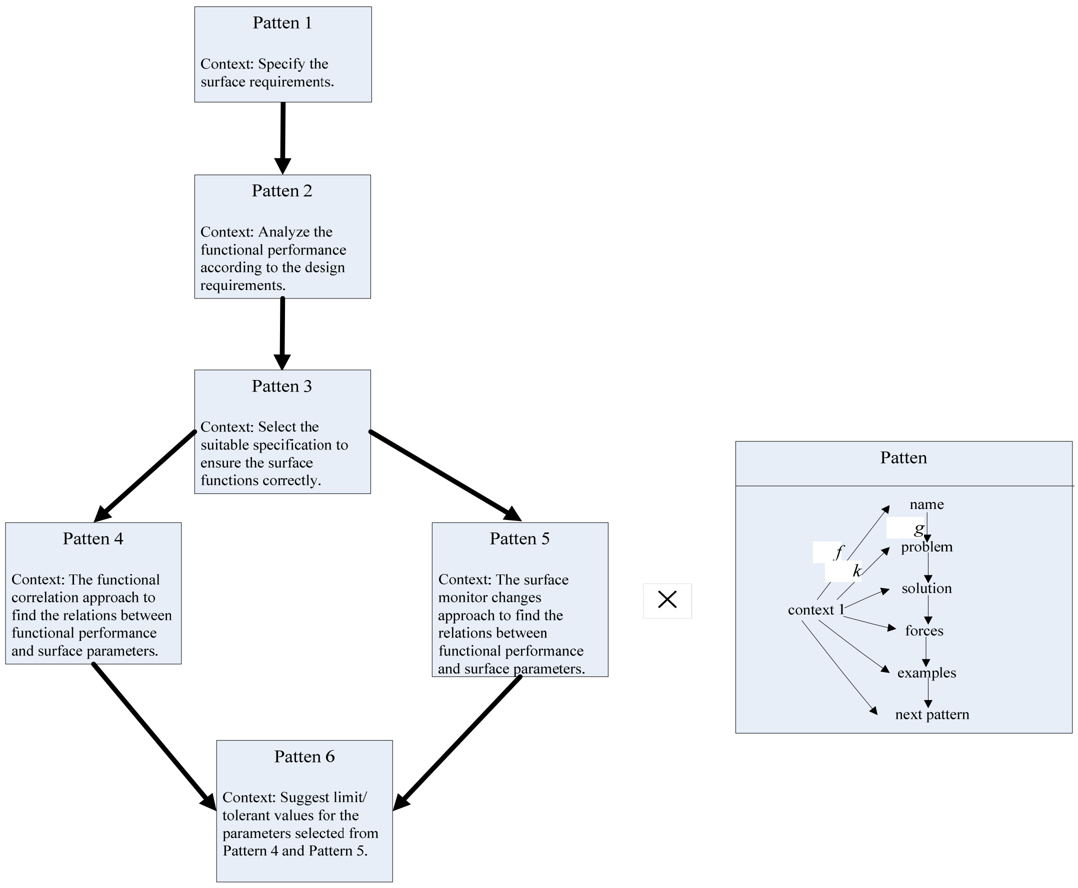

Patten 1
Context: Specify the surface requirements.
Patten 2
Context: Analyze the functional performance according to the design requirements.
Patten 3
Context: Select the suitable specification to ensure the surface functions correctly.
Patten 4
Context: The functional correlation approach to find the relations between functional performance and surface parameters.
Patten 5
Context: The surface monitor changes approach to find the relations between functional performance and surface parameters.
Patten 6
Context: Suggest limit/ tolerant values for the parameters selected from Pattern 4 and Pattern 5.
×
Patten
name
g
f
problem
k
solution
context l
forces
examples
next pattern

Figure 3.24: Product order of Function component.

Actually, Figure 3.24 is a product order which is a Cartesian product of two posets:

namely, patterns collection poset and internal object poset. As the transitivity definition of poset, all arrows among internal objects must commute (e.g. if *f*: *context1*$\rightarrow$ *name*, *g*: *name* $\rightarrow$ *problem*, so $f \circ g$ must equal to *k*: *context1*$\rightarrow$ *problem*). The Pattern 4 and Pattern 5 are two optional approaches to find the relations between function performances and surface parameters, which uses injection functors to form Pattern 6 together. The seven internal objects are key elements in the pattern language (Rising, 1998 [76]):

- **Name**: A clear format header to describe the pattern.
- **Context**: Suitable scenarios to apply the application problem.
- **Problem**: A statement of the application problem.
- **Solution**: A viable solution to the problem. Many problems might have more than one solution. The fitness of a particular solution is determined by the context of problem domain.
- **Example**: A case analogy on the problem solution.
- **Force**: There often exists contradictions when choosing a solution to a problem. Each solution is ranked with weights described by certain forces.
- **Next pattern**: Pointing to the next pattern required to form an integrated pattern language instance.

The discussion above illustrates that the Category Theory can give a complete implementable representation for a pattern language in the Function component with an open platform for GPS experts to add more knowledge in future. This pattern language can help users to find the best way to carry out their tasks with a clearly guided procedure. Moreover, users can record their valuable knowledge (e.g. a surface parameter for a specific function) within a logical linked structure.

3.7.2 Knowledge modelling in Specification Component for Surface Texture

The Specification component provides detailed geometrical specifications for the selected surface parameters including information obtained from partition, extraction and filtration operations. For example, to satisfy the functional requirements of a cylinder liner, the Function component of VirtualGPS system suggests using the surface texture parameter *Rz* with a tolerance value at 4um. The Specification component in turn recommends the complete information relating all these operational procedures such as evaluation length for extraction, and the bandwidth for filtration. Due to the complexities and intertwined attribute relationships and constraints among all viable operational procedures, Category Theory is used to model them in diagrams devised in the categorical object model. Figure 3.25 gives an example of the modeling diagram when defining a default constraint between the operations of extraction and partition.

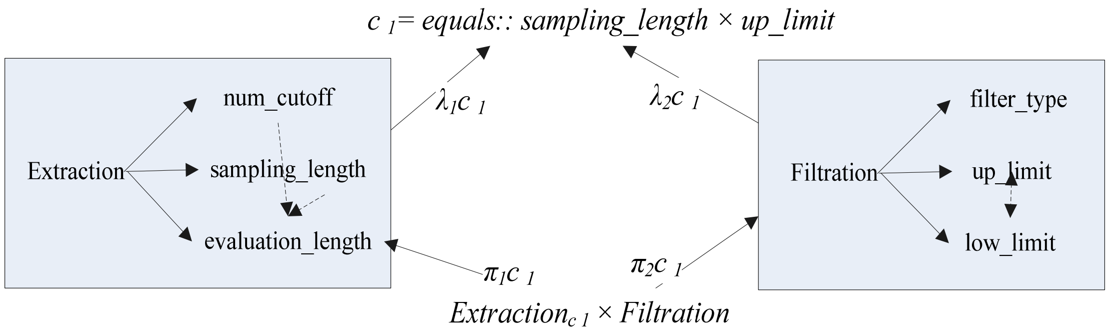


Figure 3.25: Pullback representation of the constraint "*equals*".

As shown in Figure 3.25, extraction and filtration are modelled as class categories. The c_1 demonstrates a constraint relationship between extraction and partition, which is structured by the construct of a "pullback" of Category Theory. A pullback is a product with restricted objects. In the case of Figure 3.25, the expression "*equals:: sampling_length × up_limit*" is the name and type of the pullback, where "*Extraction$_{c1}$×Filtration*" is the restricted product over c_1 (c_1 represents the restricted object – "*ExtractionToFiltration*" with restricted condition "equals" here). The notations $\pi_1 r$ and $\pi_2 r$ are projections of the product into the initial instance categories of the "*Extraction*" and "*Filtration*" respectively. While $\lambda_1 r$, $\lambda_2 r$ are represented as arrows injecting the initial instance categories into the pool of instances of this constraint relationship. The detailed explanations on the construct of "pullback", and how it can be used in representing constraints among entities, can be found in a paper published by Nelson etc. in 1994 (Nelson et al., 1994 [66]). The reason why knowledge base designers use pullback rather than universal product to represent the relationships or constraints in modelling of the GPS knowledge base is that the pullback can express stricter semantic construct for relationship linkages. The stricter semantic construct is of vital importance for knowledge base designers to clarify their design thoughts especially in a refinement design process and to communicate with other designers. In a contrast, the object-oriented database developers focus more on object-oriented development issues, so they do not need such strict semantic construct but the well defined relationship category (restricted object). The restrict object will become relationship category, the restrict condition will become methods in the relationship category, and the relationship category will be stored the same as a class category. The detailed example on how to mapping the pullbacks in knowledge base modellings into categorical products for database schema is demonstrated in Section 3.8. Figures 3.26 and 3.27 show the other two default constraints in the Specification component, which are modelled in same way as Figure 3.25.

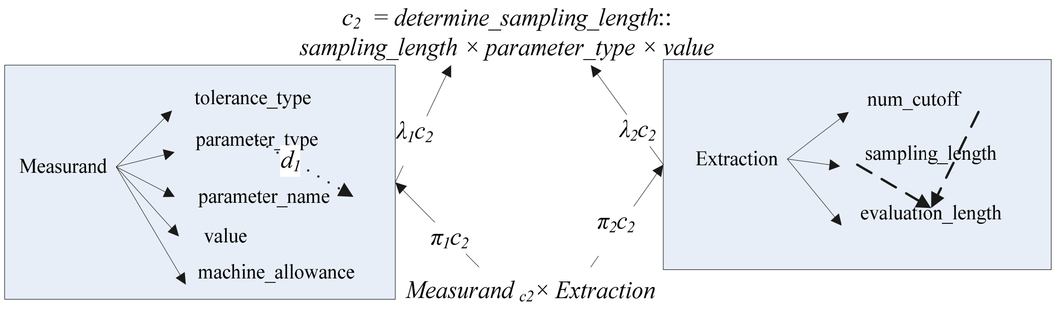


Figure 3.26: Pullback representation of the constraint “*determine_sampling_length*”.

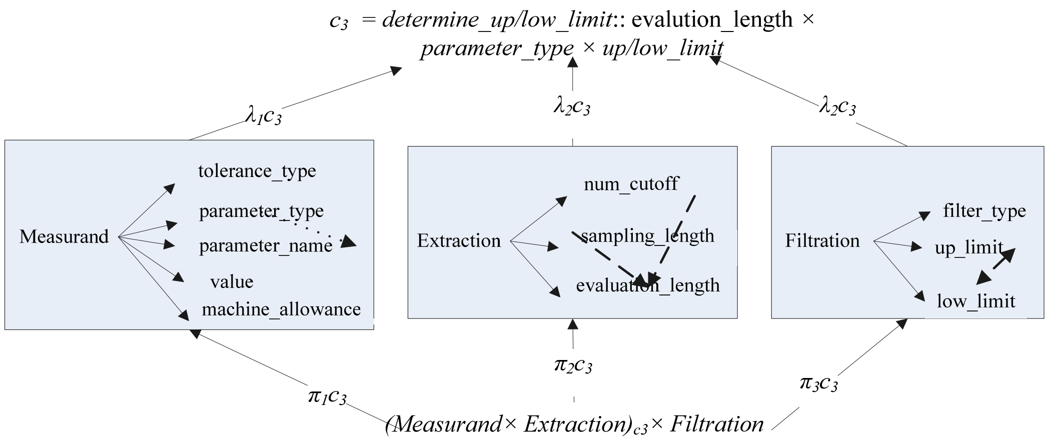


Figure 3.27: Pullback representation of the constraint “*determine_up/low_limit*”.

A higher level relationship − “Callout” is demonstrated as Figure 3.28.

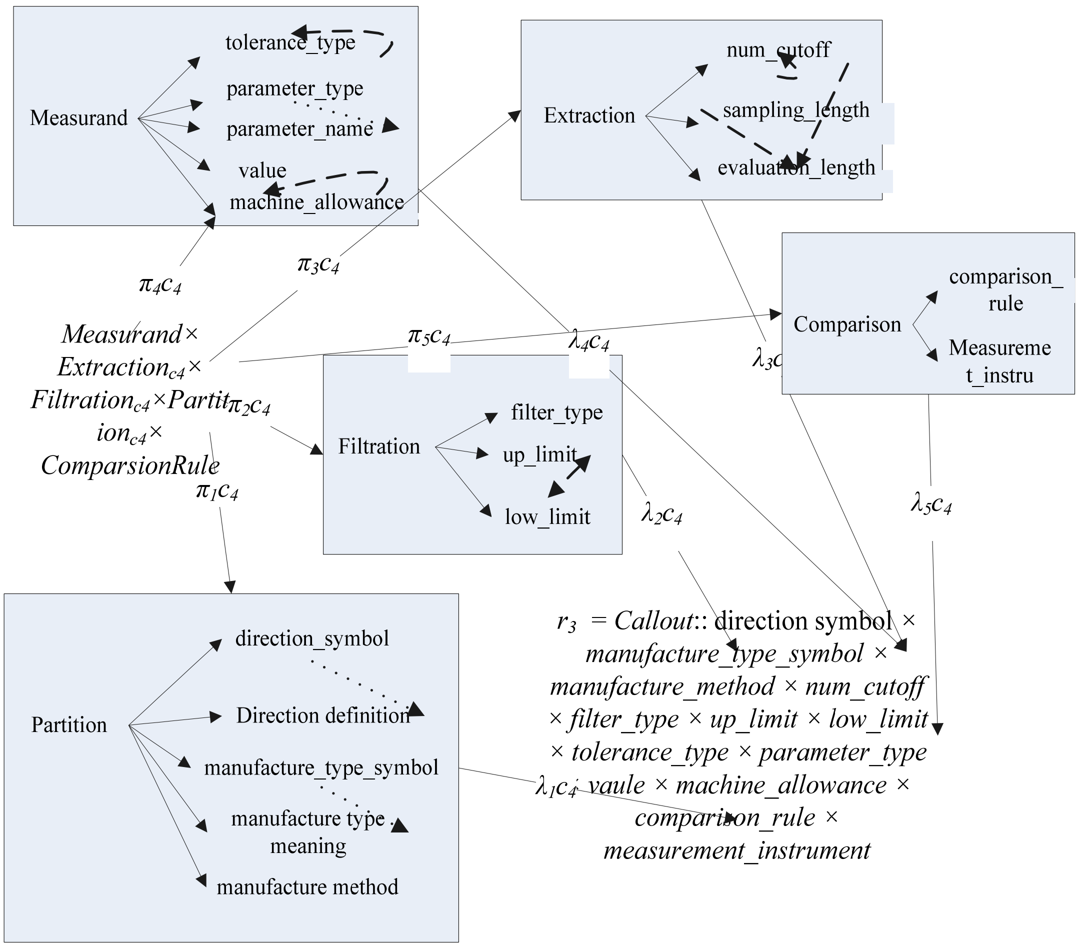


Figure 3.28: Pullback representation of the "*Callout*" relationship.

Figure3.28 shows how these three lower level constraints (Figure 3.25, Figure 3.26 and Figure 3.27) form the overall modelling of the knowledge base in the Specification component. The dashed line arrows in Figure 3.25, 3.26, 3.27, and 3.28 represent method arrows, while the dotted line arrows are functional dependency arrows between internal objects (except attribute arrows). Thus, by representing surface texture operational procedures as categories, attributes of them as internal objects, and the corresponding relationships and constraints as pullbacks between categories, the whole Specification component can be logically and structurally expressed. All arrows in Figure 3.25, 3.26, 3.27, and 3.28 must commute in a manner to ensure consistency.

3.8 Categorical Representation for a Database Schema

After describing the knowledge base in Category Theory terminology, this project moves on to the next phase of developing an innovative DBMS with the ability of fully supporting the devised categorical object model. The first step in developing this categorical DBMS is to do further refinements on these categorical object modelling diagrams devised in knowledge base design stage. Once these refined object models are established, the DBMS will have a sound mathematical foundation to ensure the integrity of the database schema when applying operations such as addition, deletion, and modification. The object models in categorical DBMS refines object models in the GPS knowledge base (see Section 3.7.2) by allowing them more computing focused from following aspects:

- As this project has chosen Java to implement the system and Java is a strongly typing language, the categorical object model for categorical DBMS should be added with a typing mechanism. The detail explanation on the typing mechanism is discussed in Section 3.4.5. The example of defining types for internal objects can be seen in Figure 3.30.
- The "pullback" construct for relationships or constraints in modelling of a knowledge base is generalized to be the "product" construct. Compared with relational algebra which defines relationship as projection or Cartesian product, the categorical representation of relationships in the categorical DBMS is the categorical product. As a product for a relationship or a constraint is mapped on the category level, it is formed by categories and functors, instead of internal objects and arrows defined in the basic definition. Moreover, the vertex of the product becomes a category – relationship category. The relationship categories are stored and managed in the DBMS in the same way as class categories and instance categories. See Figure 3.30.
- As an object-oriented DBMS assigns a unique identifier to every instance of a database entity, the vertex of the universal cone (limit) can be used to model the unique identifiers (see Figure 3.30) (Nelson and Rossiter, 1995 [77]). If we view the universal cone as a category, the vertex of the universal cone is actually the initial object in this category with an arrow from itself to every other internal object (attribute arrows) in the same category, which stores a unique automatically generating identifier values. These identifier values cannot be modified by applications at run time and they are independent of how objects are created and manipulated. By modelling the database in this style, users have been spared the task of defining keys (primary keys or candidate keys).

- The categorical object modelling diagrams in the categorical DBMS are also required to remove all transitive functional dependencies on initial internal objects in the models used in knowledge base modelling to satisfy the BCNF normal Form (except the atomic requirement in 1NF). This is achieved by removing functional dependency arrows in categories through building new lower level categories, and then linking them with their corresponding higher level categories by using "faithful" functors. The functional dependency arrows that need to be removed do not include these attribute arrows. For example, the arrow d_1 in Figure 3.26 indicates that the internal object "parameter_name" is functional dependent on internal object "parameter_type" in the class category of "*Measurand*", which again make the "parameter_name" transitive depending on the initial object of "*Measurand*". Therefore, a new class category named "*ParameterInfo*" needs to be devised and a faithful functor injects this class category into the "*Measurand*". See Figure 3.29.

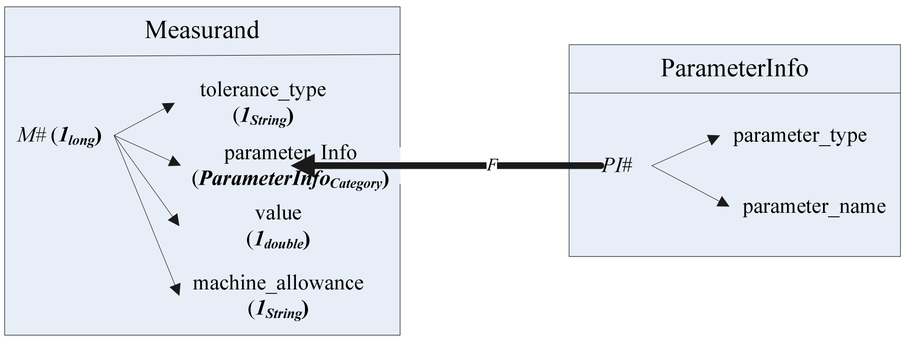


Figure 3.29: Two level class category construct.

Based on the above four points, Figure 3.30 gives an example of a refinement of Figure 3.28 for database schema modelling, which is actually a 5-ary product relationships — "*Callout :: direction symbol* × *manufacture type symbol* × *manufacture method* × *num_cutoff* × *filter type* × *up limit* × *low limit* × *tolerance type* × *parameter type* × *value* × *machine allowance*". The *P#*, *E#*, *F#, CR#* and *M#* in the diagram are unique identifiers for "*Partition*", "*Extraction*", "*Filtration*", "*Comparion*" and "*Measurand*" respectively. The F_1, F_2, F_3, F_4 and F_5 are functors that project from relationship category "*Callout*" into the five class categories. In "*Extraction*" class category, the "*evaluation length = num_cutoff* × *sampling_length*" clause indicates the two arrows (*m* and *n*) are method arrows and the other arrows are dependency arrows. In Figure 3.30, $\boldsymbol{1_x}$ indicates primitive types such as double, integer or string in Java and ***ClassName***$_x$ indicates class category types, arrow types or other complex data structure types such as Tree, List and Collection.

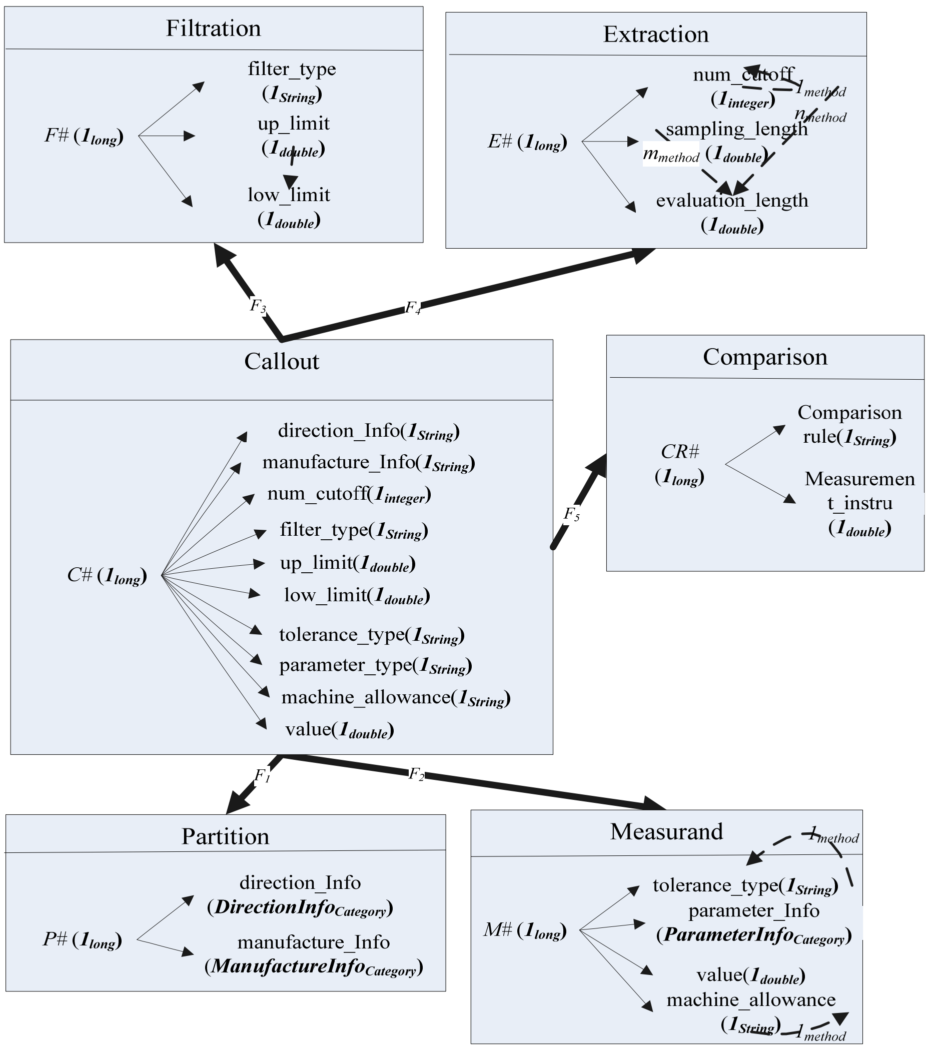


Figure 3.30: The 5-ary product relationship for the "*Callout*".

The detailed implementation explanations (inference rule specifications) for all method arrows defined in Figure 3.30 can be referred to in Section 5.3.3 of Chapter 5. Compared with Figure 3.28, Figure 3.30 is more structured and computing oriented, which focus on objects and relationships or constraints among these objects. However, Figure 3.28 is more semantic oriented and focus more on system logics and rules.

3.9 Summary

This chapter illustrates how to use Category Theory to model the whole VirtualGPS system with a set of detailed examples. This chapter also proves that the Category Theory can serve as a formal mathematical basis for object-oriented knowledge applications.

CHAPTER 4 IMPLEMENTATION OF THE CATEGORICAL DBMS

This chapter records in detail the implementation of the categorical DBMS. The implementation includes discussions on how the DB4O (Database for Objects) was chosen to be a basis for the implementation, the categorical ODMG architecture for the categorical DBMS, extensions on the physical level of the DB4O, and how to implement the categorical object model on it.

4.1 Basic Criteria for Implementation

By making use of the basic features of existing object-oriented DBMS products such as physical storage mechanism, indexing strategy and transactional controller etc., the so-called “categorical” DBMS based on Category Theory can be developed in an efficient manner. As Table 2.9 shown in Section 2.2.4.2 of Chapter 2, it is legitimate to choose the Objectivity/DB as basis to develop the categorical DBMS for its ODMG standard compliance. However, this project has chosen the DB4O to form the internal level of the categorical DBMS for the following reasons:

4.1.1 Conformability

As Table 2.9 highlighted, there are currently no matured DBMSs that can fully support ODMG 3.0. Figure 4.1 illustrates the architecture of ODMG 3.0.

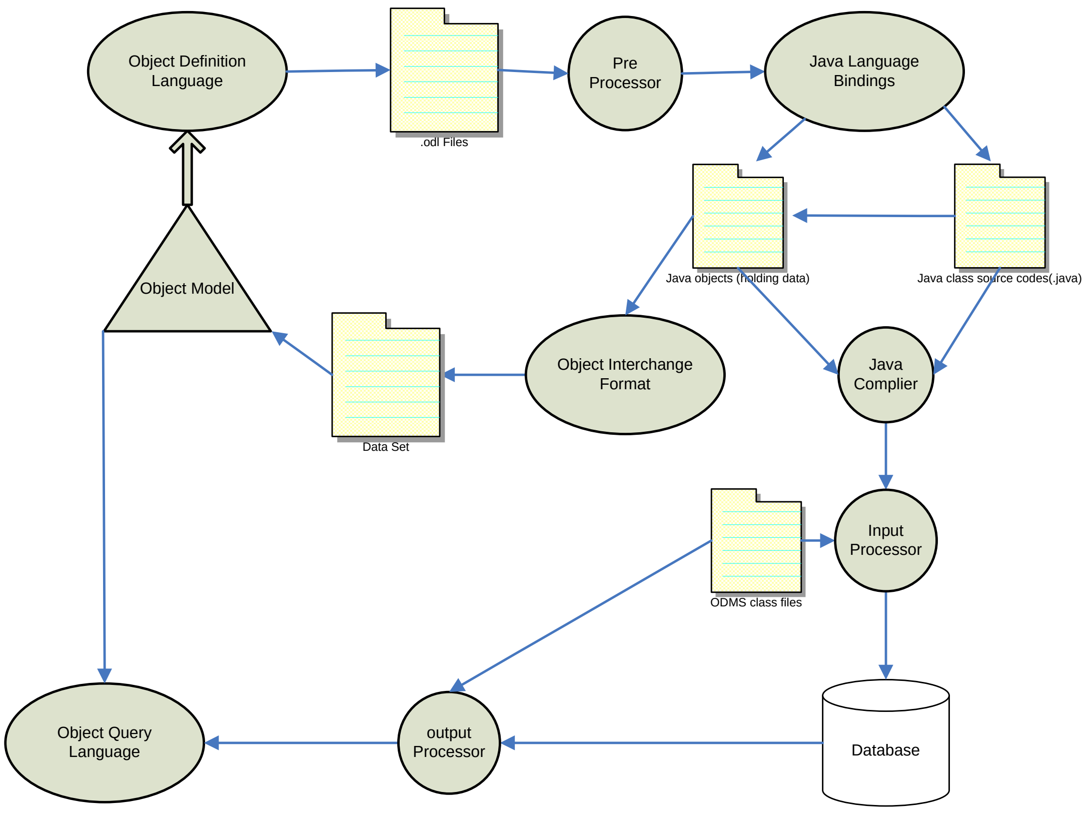


Figure 4.1: The architecture of ODMG 3.0.

Figure 4.1 also highlighted the development process of an ODMG compliant Object-oriented Database Management System (ODMS): first, database designers start with representing object models in a diagrammatic way (e.g. E-R diagram) and then using an Object Definition Language (ODL) to translate the diagrammatic model into the programming language independent schema codes. The ODMG compliant ODMSs should offer a pre-processor that can automatically generate source codes in the form of Java class declarations according to the schema code generated. After database programmers bind the detailed implementation codes (methods) into these Java class declarations, a set of objects holding real data will be created on these Java classes, and both Java classes and their objects will be compiled by a standard Java compiler. If required, objects can be saved into a file following an Object Interchangeable Format (OIF) to enable data sharing in different ODMSs. Finally, the input processor will link necessary ODMS control files for maintaining objects storing in the underlying database. The ODMS standard also defined an Object Query Language (OQL) based on the SQL-92, with the output processor in charge of translating the OQL into internal codes understandable by the ODMS.

However, there are three main obstacles in this architecture design:

- **Object Model Inefficiency**. The ODMG 3.0 did not define any specific diagrammatic facility to represent its object model. Most of the ODMG compliant ODMSs use E-R or E-R extension diagrams to visualize representations of applications. Moreover, the object model defined in ODMG suffers from the same drawback as other object-oriented data models due to the weak mathematic foundations and lack of semantic constructs to support new applications. For example, the object model in ODMG can only support the definition of binary relationships, without supporting n-ary relationships. In this project, Category Theory was adopted to model the complex relationships and constraints among GPS standards, which avoided any mapping code generation between the data in a database and the data from an application through using the same model mechanism in both sides. Thus an object-oriented DBMS that can fully support the categorical object model is required in this project.
- **Intrinsic Implementation Obstacles**. Some theoretical points defined in the ODMG standards are difficult to be implemented in ODMSs. For example, it is challenging to build a pre-processor that can fully support automatic translating of ODL into object-oriented languages. In Objectivity/DB, the ODL is actually the standard C++ 3.0 language with extensions to support persistence-capability and object associations (Objectivity, Inc., 2006 [78]). Therefore, although several current mainstream ODMSs claimed to be ODMG compliant, in reality they fall short of the bar.
- **Out of date of OIF**. The latest version of ODMG was defined in 2000. Almost at the same time, a "new" universal data interchange technique −XML was published (XML 1.0) in 1998 (Harold, 2004 [79]). Since then, XML has become the most studied and adopted standard for describing structured data to be exchanged between applications (e.g. database application), especially acrossing the global Internet and World Wide Web (WWW) (Harold, 2002 [80]).

4.1.2 Compatibility

Based on the discussions above, this project needs to devise a new object model which can represent all potential constructs required in the GPS knowledge base; to develop a pre-processor to translate the extended ODL codes into standard Java classes; and to apply XML to substitute the obsolete OIF defined in the ODMG for data exchanges. This in turn required the internal programming routines of the selected DBMS to be updated accordingly. Although, Objectivity/DB and Versant are products developed by large corporations, with lots of extra functions for helping users to create and manage their databases. They are both pre-packed and do not allow changes to their internal codes. Moreover, the costs of both products were also prohibitive for such a research project. Another open source object-oriented DBMS evaluated − Ozone − is lack of application and learning supports, which have not been improved since Ozone 1.2 released in 2004.

4.1.3 Robustness

DB4O is an open source native object database for Java and .NET, which can support all features defined in the first manifesto that an object-oriented DBMS must include and should include (Db4objects, Inc., 2007 [81]). The DB4O can be directly embedded in the host Java or .NET applications without requiring any extra installations or setups on local platform in advance. It has small memory foot-print (500kb library) while supporting object caching, native garbage collection, ACID transaction (Atomicity, Consistency, Isolation, and Durability), client/server architecture, automatic management and versioning of database schema (ETH, 2007 [82]). The DB4O provides the General Public License (GPL), which enables easy download, studying, evaluation and usage of DB4O in GPL compliant projects. Moreover, DB4O members can get free developer licenses to contribute to the DB4O's ongoing developments. It was decided in this research that the DB4O is a suitable tool and template for implementing the categorical object model devised in this project. A detailed introduction of DB4O will be presented in Section 4.2.

It was concluded based on the above considerations that this project decided to produce an ODMG 3.0 extension named "categorical ODMG" for forming the following architecture as illustrated in Figure 4.2.

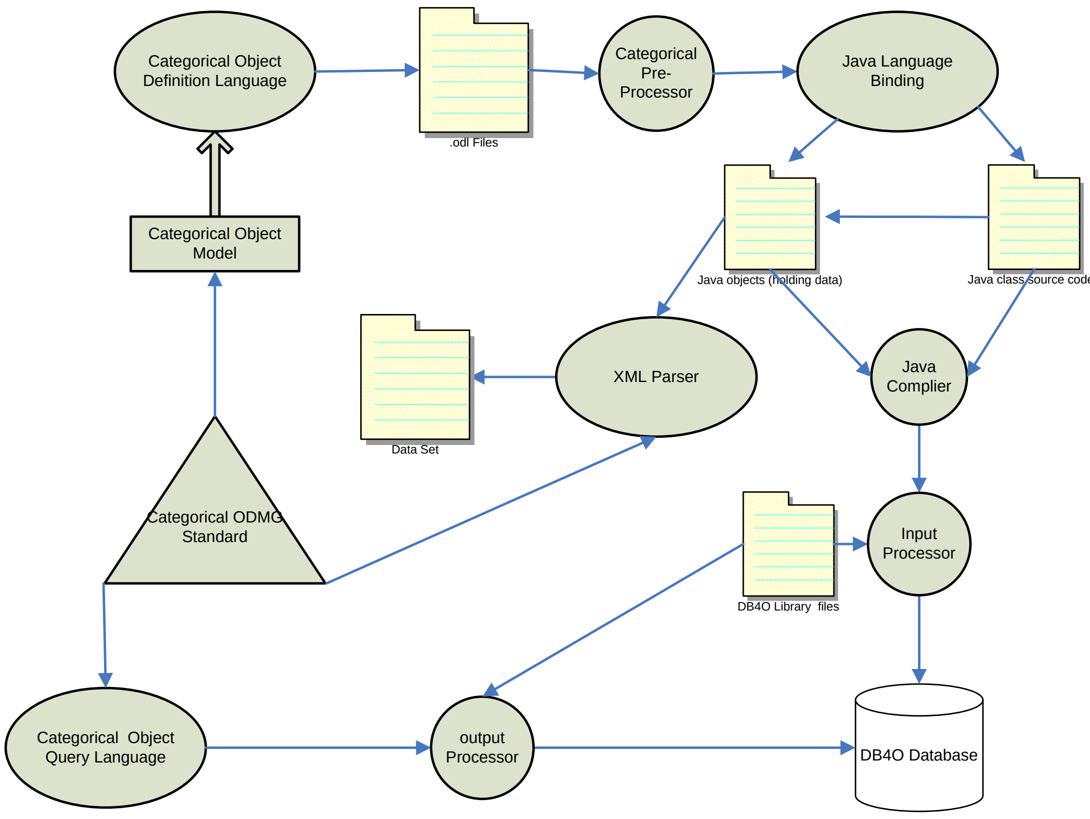


Figure 4.2: The architecture of categorical ODMG.

The customised categorical ODMG extends the ODMG 3.0 with a categorical object model, an extended ODL which supports all semantic constructs of the categorical object model (e.g. definition of n-ary relationship and auto-persistent definition), a series of Java binding Application Program Interfaces (APIs) to support categorical object model, and a categorical OQL based on functor mappings and compositions.

4.2 Native Design of DB4O

Built on new object database technology, DB4O is currently the only viable database that is compatible to both Java and .NET for providing cross-platform portability that liberates users from proprietary vendors' high licensing fees (Db4objects, Inc., 2008 [83]). DB4O (database for objects) was developed by Db4objects, Inc., which is a privately-held company based in San Mateo, California (Paterson, 2006 [84]). It was firstly created by the chief software architect Carl Rosenberger and shipped in 2001. More than one hundred commercial and private pilot customers formed a loyal user community that endorsed the DB4O from its earliest days and proved it ready for mission-critical applications prior to its commercial launch in 2004. The visions of DB4O can be summarized as followings (Db4objects, Inc., 2005 [86]):

- Development of a lightweight, apt object-oriented persistence solution with the availability of a popular, affordable, embeddable and open source.
- Becoming the mainstream persistence architecture on all mobile and embedded devices running on Java or .NET.
- Achieving consolidation in a market that overruns with hundreds of vertical niche vendors offering predominantly outdated or unsuited pre-relational or relational technology at exorbitant prices at present.

The DB4O provides a wide array of unique, object-oriented database functionalities by harnessing the benefits of object-oriented programming languages: seamless object-oriented storage (store any complex object with just one line of code); object-oriented replication (dRS); and object-oriented queries (e.g. Native Queries, and Simple Object Database Access (SODA)). The core features of DB4O can be summarized as followings (ETH, 2007 [82]; Paterson, 2006 [84]):

- No requirement on data conversion or mapping (directly object storage support)
- No changes required to classes to make objects persistent
- Single line of code to store objects of any complexity and persistence by reachability
- Embeddable to large and complex systems
- Support Java generics
- ACID transaction support
- BTree index support
- Client/Server support
- Automatic management and versioning of database schema
- Object caching and integration with native garbage collection
- Seamless Java or .NET language binding
- Native Queries/SODA
- Portability and cross-platform deployment

Based on these core features, the architecture of DB4O can be illustrated as in Figure 4.3:

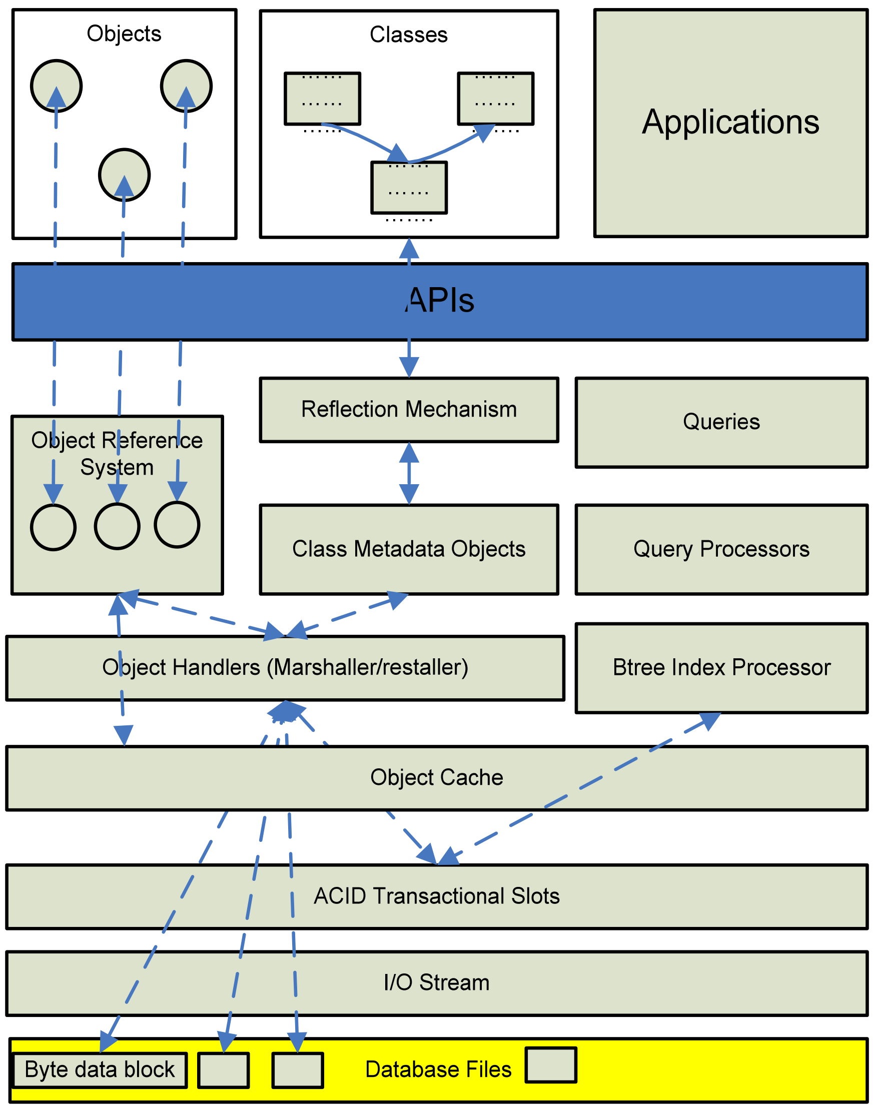


Figure 4.3: System architecture of the DB4O.

In comparison with closed-source products such as Objectivity/DB and Versant, the DB4O has five distinctive advantages:

1. DB4O is an open source database with small library files (500k). Database developers in this project can study its structure and make necessary changes to support the categorical object model.
2. DB4O focuses on the embedded and portable database market driven by object-oriented programming environments. For example, a German company − Mobilanten gained a competitive edge in its new product range by providing a Personal Digital Assistant (PDA)-based solution for field workers of mid-sized utilities using DB4O, whereas competitors using the relational DBMS required bulky laptops to process assets, orders, and customer information (Replicating some 300,000 objects was just not feasible using relational

databases on a PDA, while synchronizing objects via DB4O proved to be extremely efficient) (Db4objects, Inc., 2005 [85]). Through embedding the DB4O core, this project can be moved to any compatible platforms without requiring complex installation procedures.

3. DB4O is quicker at runtime than other bulky object-oriented DBMSs. Various tests run by INDRA Sistemas's new real-time control system on the Spain's high speed bullet train, the AVE, had shown that no other systems except the DB4O can handle the huge load of processing over 200,000 heterogeneous objects per second (Db4objects, Inc., 2005 [85]).
4. ACID transaction support.
5. Professional and stable user community. DB4O has more than 35000 registered users up to 2008 who are contributing to DB4O's online community for its further development (DB4O Developer Community, 2008 [86]).

In addition to adding codes for supporting the categorical object model on the DB4O, the categorical DBMS also provides five extensions on the physical level of DB4O, which enabled the DB4O customisation and superior performance for the VirtualGPS system in this research. The five extensions will be illustrated in section 4.3.

4.3 DB4O Customisation

The five extensions of DB4O supported by the VirtualGPS development in this research are: the extension of the Simple Object Database Access (SODA) in DB4O to support functor mappings and compositions, the extension for supporting automatic persistence, the extension for supporting the storage of physically clustered objects, the extension for supporting referential integrity checking, and the extension for supporting the categorical Object Definition Language (ODL). The detailed demonstration of the extension of the SODA to produce the categorical manipulation language in this DBMS can be found in sub-section 4.4.1.5. The other four are discussed in Section 4.3.1, 4.3.2, 4.3.3 and 4.3.4 respectively.

4.3.1 Automatic Persistence

This means persistence capabilities can be automatically granted to instance categories of class categories that extends the "*PersistCategory*" class category. The purpose for adding this feature into the categorical DBMS is to support the automatic result storage: all instance categories extending the "*PersistCategory*" class category generated during a categorical query process will be automatically stored back to the database (e.g. the "*Callout*" class category as example).

4.3.2 Physically Clustered Objects Storage

This extension enables a group of related instance categories to be stored physically together in the categorical DBMS by using the "*ClusterContainer*" class. The "*ClusterContainer*" has a "*cluster (Category category)*" method to add an instance category into a cluster. The main motive to do this is to let the categorical DBMS to retrieve, delete or update a group of related instance categories quickly and efficiently.

4.3.3 Referential Integrity

Referential Integrity is added into DB4O to check whether an instance category is referred by other instance categories or not. A special byte will be added to the storage schema on every instance category to record its reference number.

4.3.4 ODL support

The ODL used in this categorical DBMS is based on the ODL defined in ODMG 3.0 with extensions to support automatic persistence capability, n-ary relationship definitions, and arrow mappings etc., in the forms shown below:

```
class ExtractionToFiltration : extent PersistCategory {
      attribute double sampling_length;
      attribute double up_limit;
      relationship {
           ary =2;
           functor1 = <Extraction::sampling_length(1)> // 1:1 relationship
          functor2 = <Filtration::up_limit(1)> // 1:1 relationship
      }
}
```

List 4.1: ODL definition for "*ExtractionToFiltration*" class category.

Once the ODL schema conforms to the diagrams of the object model being specified, it needs to be validated against the categorical ODL specification to determine if the syntaxes in the ODL schema are correct. The Java Compiler Compiler (JavaCC) tool performs this syntax checking. JavaCC is a parser generator designed for using with Java applications (Java Net, 2007 [87]). JavaCC reads a grammar specification and generates a parser (Java program) that is used to recognize matches to the specified grammar. This parser is specified in a file with a '*.jj*' extension. For example, after the "*ExtractionToFiltration*" has been correctly parsered by JavaCC, the categorical DBMS will automatically generate an "*ExtractionToFiltration*" Java class (class category). The Table 4.1 shows the mapping relations from ODMG collection data types in ODL to Java classes defined in the categorical DBMS.

ODMG Interface	Java Bindings	Categorical database Implementations	Description
DArray	java.util.List	CTArray	Ordered, fixed size
DList	java.util.List	CTTree	Ordered, variable size, used to add/remove instance categories(Java objects)
DSet	java.util.Set	CTSet	Unordered, no duplications, used to check duplicated instance categories
DCollection	java.util.Collection	CTCollection	Ordered, fixed size

Table 4.1: The Java binding APIs in the categorical DBMS.

The next section focuses on discussing processes to implement the categorical object model on the DB4O.

4.4 Implementation of the Categorical DBMS

The categorical DBMS devised in this research satisfies the classic three-level architecture defined in ANSI/SPARC: a physical database level, a conceptual database level and an external database level (Tsichritzis and Klug, 1978 [88]). Figure 4.4 shows the architecture of the categorical DBMS.

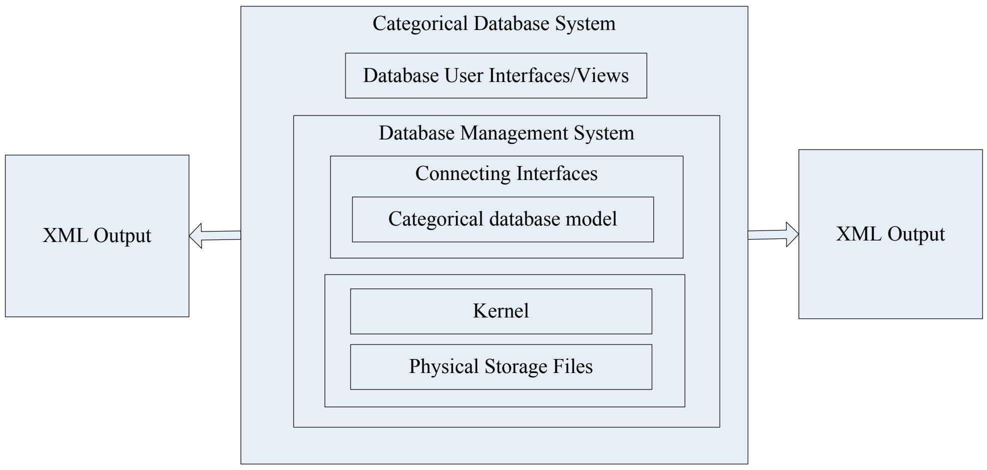

Categorical Database System
Database User Interfaces/Views
Database Management System
Connecting Interfaces
Categorical database model
Kernel
Physical Storage Files
XML Output
XML Output

Figure 4.4: The architecture of the categorical DBMS.

As Figure 4.4 demonstrated, the physical database level of the categorical DBMS contains a set of physical storage files and a kernel (file manager), which is in charge of controlling the physical data storage, building physical storage schemas and handling manipulations. The categorical object model is resided on conceptual database level, which is responsible for describing the problem domain (database schema) and to specify what needs to be stored in the database while ensuring the database integrity at all times. The conceptual database level also contains a collection of connecting interfaces which are responsible for translating physical raw data into conceptual objects, controlling concurrent accessing and passing objects and their relationships or constraints to user interfaces. The categorical DBMS contains two kinds of data: metadata and application data. Besides performing queries on application data, the metadata (database schema) itself can also be queried for information such as get the details of a class category defined in a database schema including the class name, its field names, its field types, number of its instance categories and so on. The external database level can display appropriate data on the user interfaces for different users. This DBMS can also generate XML reports to record and demonstrate querying results, which supports internet and multi-user applications through a unified consistent view port for the data crossing through whole manufacturing enterprise. The Section 4.4.1 discusses the implementation of the categorical object model following the specification of the object model discussed in section 3.4.

4.4.1 Realizing the Categorical Object Model

The categorical DBMS in this research is a compact and autonomous object-oriented database management system implemented by pure Java language. As with all main stream relational DBMS products, it also contains a data model, a database entity definition language, a database entity query language, the physical storage and retrieving mechanism, and a small visual management software package. The database entity definition language and the query language were both developed using Java to support semantic constructs of the categorical object model. The database entity definition language contains a set of Java classes (e.g. "*Category*", "*Pullback*" and "*Functors*") to create database entity specifications. These Java classes are basically organized by Java object operations such as accessor and mutator. An accessor examines the state of an object but does not change it. It typically returns a result in a pre-defined form (Sun Developer Network (SDN), 2005 [89]). Accessors are often call "*getters*", and their names often start with a "get". A Java method that changes the state of an object is called a mutator (Sun Developer Network (SDN), 2005 [89]). Mutators typically do not return a result (are declared to return "void"), although some mutators can both change state and return a result (e.g., *nextToken* in *StringTokenizer*). Mutators are often called "*setters*", which just change state without looking at the current values, and their names often start with a "set". Accessors and mutators will increase the overhead on an application programme, but are usually trivial, especially when compared with other factors, such as questionable database designs (Ambler, 2000 [90]). Accessors and mutators can improve the maintainability of the object model in the following ways for this research (Ambler, 2000 [90]):

- To provide a single point for updating instance categories (Java objects). Parent applications can only have controlled points for updating each attribute, making it easier to modify and to test. Thus, internal objects in instance categories (attribute values in Java objects) are safely encapsulated.
- To enable constraint encapsulation. For example, in a category, values of some internal objects may be constrained by values of other internal objects. These constraints can be defined in mutators using bulk setter method (update several attributes at once to keeping constraints among them). Thus, the constraints will be automatically enforced when programmers set values of the constraint internal objects. Moreover, if a critical constraint should be enforced on a internal object, such as “a value must less than 10.0 µm”, then a logical place to put this clause will be a mutator (setter).
- To enable change encapsulation. If the business rules pertaining to several attribute changes, accessors and mutators can both be potentially modified to respond to the business rules.
- To reduce coupling cost between a subclass and its superclasses. Accessors reduce the risk of the fragile base category problem where changes in super categories ripple throughout its sub categories.

Therefore, accessors and mutators are widely used in this project to reduce coupling of the database with its host application. The remainder of this section focuses on discussing the implementation of the categorical object model for this categorical DBMS — an object-oriented DBMS with a formal object model, as well as the object definition and query languages. Some fragments of the actual Java codes are also demonstrated for explaining the function tasks and forms.

4.4.1.1 Complex classes and objects

The definitions of database entities (class categories) are treated as subclasses of a Java class named “*Category*” — the base class category. The “*Category*” class contains an object instance ID and a class category ID. These two unique internal IDs are assigned automatically by the categorical DBMS using the physical storage addresses of the class category metadata and its instances in the physical storage files. All internal objects of a category can then become the "target" of arrow attributes defined in a class category. As specified in Section 3.4.1, each subclass of the "*Category*" (class category) should hold a collection of arrows where these arrows can either represent behaviors (methods) or associations (dependencies). For dependency arrows, the "*Arrow*" class is developed for holding a dependency between two internal objects in a category, for example, arrows from the unique object instance ID to other internal objects in a class category. There are two kinds of behaviour arrows – the ones between internal objects in the same class category and the others between different class categories. These two kinds of behaviour arrows will both become methods of the corresponding Java classes. The main difference between these two is when operating in the same class category, the behaviour arrows will become methods in this class category but when crossing different class categories, a class category extending the "*Product*" class category will be defined and the behaviour arrow will become a method in this subclass. As discussed in Section 3.4.1, the creation of an instance on a $\boldsymbol{CLS}_i$ is actually assigning a functor from a class category $\boldsymbol{CLS}_i$ to an instance category $\boldsymbol{OBJ}(j)_{CLS_i}$. This functor is implemented by the "*new*" operation in Java, which can automatically maintain a relation between the object instance and its belonging class. Java can automatically check whether objects are conforming to the definitions of their classes (e.g. values are in correct types, a method has correct parameters).

```
package cpt.ctdb.dataModel;
import java.lang.reflect.*;

public class Category {
        private int classInternalId;
        long objectInternalId;
        String name;
        Arrow[] arrows;

        public void setClassInternalId(int classInternalId){
            this.classInternalId=classInternalId;
      }

        public void setObjectInternalId(long objectInternalId){
             this.objectInternalId = objectInternalId;
        }

      public void setName(String name){
             this.name = name;
        }

        public void setArrowSources(Object obj,int internalID){
                Field[] fields=obj.getClass().getDeclaredFields();
                      for(int i=0; i<fields.length;i++){
                             if((fields[i].getType().getName()).endsWith("Arrow")){
                                try{
                                ((Arrow)fields[i].get(obj)).
                                setSource(Integer.valueOf(internalID));
                  }catch (Exception e){
                                          e.printStackTrace();
                                }
                             }
                      }
          }

          public void addArrows(Arrow[] arrows){
                this.arrows = arrows;
          }

          public int getClassInternalId(){
              return this.classInternalId;
         }

          public long getObjectInternalId(){
               return this.objectInternalId;
          }

         public String getName(){
              return this.name;
          }

          public Arrow[] getArrows(){
                      return this.arrows;
          }
}
```

List 4.2: Java codes for "*Category*" class category.

As demonstrated in the List 4.2, the "*Category*" class contains a set of interface methods to "set" and "get" unique internal IDs. The kernel part of the categorical DBMS contains codes which can automatically generate unique internal IDs and assign them to instance categories through the "*setClassInternalId()*" and "*setObjectInternalId()*" interface methods of "*Category*" class. Moreover, a "*setArrowSources()*" method is also devised here to set the source (unique object instance IDs as discussed before) to the "source" property of every dependency arrow.

```
package cpt.ctdb.dataModel;
public class Arrow {
     String name;
     Object source;
     Object target;

     public void setName(String name){
          this.name=name;
    }

     public void setSource(Object source){
           this.source = source;
```

```
        }

        public void setTarget(Object target){
                this.target = target;
        }

        public String getName(){
              return this.name;
        }

        public Object getSource(){
                return this.source;
        }

        public Object getTarget(){
                return this.target;
        }
}
```

List 4.3: Java codes for "*Arrow*" class category.

As shown in the List 4.3, the "*Arrow*" class contains the source and target internal objects and a unique name for this arrow. The "*Arrow*" class is used to record dependency arrows in the categorical object model.

4.4.1.2 Relationships

As discussed in Section 3.4.6, if relationships occurred at the categorical level, they are represented as categorical products between categories with consistency checks such as cardinality, as well as the membership of a product, in terms of epimorphisms and monomorphisms. The following code snippet in List 4.4 shows an example of defining a product relationship.

```
public class Product{
        String name;
        int ary;
        Object vertex;

        public void setName(String name){
              this.name=name;
        }

        public void setAry(int ary){
              this.ary=ary;
        }

        public void setVertex(Object vertex){
              this.vertex = vertex;
        }

        public String getName(){
              return this.name;
        }

        public Object getVertex(){
              return this.vertex;
        }

        public CTDBObjectSet getAllInstances(Class a_class){
            //get all instance categories for a class or a relationship
            //category.
        }
    }
```

List 4.4: Java codes for "*Product*" class category.

The "*Product*" class implements the product construct in the categorical object model, which contains the name and ary number of the product. The "*vertex*" is an instance category that holds all information of the relationship. The process of creating a 4-ary product relationship "*ProductForCallout*" in the DBMS is illustrated as following:

- Creating class categories that participate in this relationship. An example of creating "*Measurand*" class category is shown in Appendix *A*, where "*Filtration*", "*Extraction*", and "*Partition*" are created in same way as "*Measurand*". The "*Measurand*" class must be extended from the "*Category*" class defined in Section 4.4.1.1 to enable the DBMS to treat it as a category.
- Creating a class category — "*Callout*" — to be the "*vertex*" in a "*Product*" class. The "*Callout*" class must be extended from the "*PersistCategory*" class. The "*PersistCategory*" enables the "*Callout*" with automatic storage capability. Appendix *B* shows the detail of its implementation.
- Creating a class category "*ProductForCallout*" that is extended from the "*Product*" class category defined in section4.4.1.2 to allow the DBMS dealing with it as a product relationship category. Appendix *C* highlights the detail of this process. The categorical DBMS offers methods for users to check the types of functors involved in the product construct through verifying the cardinality and membership of each class category participating in this relationship linkage. These "*checkXXX()*" methods in the class category "*ProductForCallout*" carry out the checks for the cardinality and membership of "*Measurand*" class category. The detailed definition of the "*Functor*" class category will be explained in Section 4.4.1.4.
- Creating instances for class categories defined above and using them to populate the "*ProducForCallout*" class category. Once the instances for "*ProductForCallout*" are created, they will be stored in the database. A code snippet of this process is listed in Appendix *D*:

4.4.1.3 Encapsulation

The categories in this project encapsulate all the relevant attributes and operations together as a collection of internal objects and arrows among them. Every class category has been implemented as a Java class, where the internal objects become attributes of a class category in appropriate data types, and the method arrows become methods with correct parameters. All the properties of a category can be referenced or navigated through using Java's native reference and name space mechanisms. The internal unique identifier (initial internal object) can be used to distinguish a instance category from others created on the same class category.

4.4.1.4 Functors and Natural Transformations for Comparison Processes

In this project, functors and natural transformations were mainly used to model comparison operations for every stable measurement procedure in the verification stage. To implement these comparison operations, the following steps had been taken:

- Devising a "*Functor*" class to hold arrow mappings from one category to another and an "*ArrowMapping*" class to record the details of an arrow mapping, as shown in List 4.5.

```
import cpt.ctdb.CTCollection;
public class Functor {
    String name;
    Category source;
    Category target;
    CTCollection arrow_mappings;

    public void setName(String name){
        this.name = name;
    }

    public void setSource(Category source){
        this.source = source;
    }

    public void setTarget(Category target){
        this.target = target;
    }

    public void setArrowMappings(CTCollection arrow_mappings){
        this.arrow_mappings = arrow_mappings;
    }

    public String getName(){
        return this.name;
    }

    public Category getSource(){
        return this.source;
    }
```

```
        public Category getTarget(){
                return this.target;
        }
}

public class ArrowMapping {
      Arrow source_functor;
      Arrow target_functor;

      public void setFunctorSource(Arrow source_functor){
            this.source_functor = source_functor;
      }

      public void setFunctorTarget(Arrow target_functor){
              this.target_functor = target_functor;
      }

      public Arrow getFunctorSource(){
              return source_functor;
      }

      public Arrow getFunctorTarget(){
              return target_functor;
      }
}
```

List 4.5: Java codes for "*Functor*" and "*ArrowMapping*"class categories.

- Devising a "*MeasurandForComparison*" class to hold all measurands and their corresponding information for every measurement procedure, as displayed in List 4.6.

```
import cpt.ctdb.CTTree;
import cpt.ctdb.dataModel.*;

public class MeasurandForComparison extends CTTree{

      public Arrow interObjId_id;
      public Arrow interObjId_measurandType;

      public void setArrows(Arrow interObjId_id, Arrow
                              interObjId_measurandType){
            this.interObjId_id= interObjId_id;
            this.interObjId_measurandType= interObjId_measurandType;
      }

      public void setTargetForIdArrow(int id){
            this.interObjId_id.setTarget(Integer.valueOf(id));
      }

      public void setTargetForMeasurandTypeArrow(String
                                                      meaurandType){
           this.interObjId_id.setTarget(meaurandType);
      }
}
```

List 4.6: Java codes for "*MeasurandForComparison*" class category.

- Devising a "*Value*" class to hold the suggested tolerance values or measured values as shown in List 4.7

```
public class Value extends CTTree{
        public Arrow interObjId_id;
        public Arrow interObjId_Value;

       public void setArrows(Arrow interObjId_id, Arrow
                                  interObjId_Value){
            this.interObjId_id= interObjId_id;
            this.interObjId_Value= interObjId_Value;
       }

       public void setTargetForIdArrow(int id){
            this.interObjId_id.setTarget(Integer.valueOf(id));
       }

      public void setTargetForValueArrow(double meauredValue){
            this.interObjId_id.setTarget(Double.valueOf(meauredValue));
       }
}
```

List 4.7: Java codes for "*Value*" class category.

- Devising a "*NaturalTransformation*" class to implement mappings between functors, as shown in List 4.8.

```
public class NaturalTransformationSquareMaps {
       Category source_functor_left;
        Category source_functor_right;
        Category target_functor_left;
        Category target_functor_right;

     public void setSourceFunctorLeft(Category source_functor_left){
            this.source_functor_left = source_functor_left;
     }

     public void setSourceFunctorRight(Category source_functor_right){
           this.source_functor_right = source_functor_right;
      }

     public void setTargetFunctorLeft(Category target_functor_left){
            this.target_functor_left = target_functor_left;
      }

      public void setTargetFunctorRight(Category target_functor_right){
          this.target_functor_right = target_functor_right;
     }
     ……………//getter methods
}

public class NaturalTransformation {
       String name;
        Functor source;
        Functor target;
        List NaturalTransformationSquareMaps;

     public void setName(String name){
            this.name = name;
      }
```

```
        public void setSource(Functor source){
              this.source = source;
          }

         public void setTarget(Functor target){
            this.target = target;
         }

         public void setNaturalTransformationSquareMaps(List
             NaturalTransformationSquareMaps){
                     this.NaturalTransformationMaps =
                                             NaturalTransformationMaps;
        }

        public String getName(){
             return this.name;
          }

          public Functor getSource(){
              return this.source;
        }

          public Functor getTarget(){
              return this.target;
        }

          public List getNaturalTransformationMappings(){
              return this.NaturalTransformationMappings;
          }
}
```

List 4.8: Java codes for "*NaturalTransformation*" class category.

The "*NaturalTransformationSquareMaps*" class was used to ensure the consistency and commutations of the natural transformation square as shown in Figure 3.22 in Chapter 3. The "*Comparison*" class category uses "*NaturalTransformation*" to implement comparisons in a stable measurement procedure. The "*Comparison*" class category offers comparison methods in form as shown in List 4.9.

```
        public int compare() {
          List temp = this.getNaturalTransformationSquareMaps();
          For(int i=0;
               i<temp.size();i++){           If(!((Double)this.interObjId_
               measuredValue.getTarget()).compareTo((Double)((( Com
               parisonValue )(NaturalTransformationSquareMaps)
               temp.get(i)).getTargetFunctorRight()).getValueArrow().ge
               tTarget()){
               return false;
          }
    }
```

List 4.9: Java codes for "*Compare()*" method.

4.4.1.5 Query Formations

The query strategy defined in the VirtualGPS system uses functors to map from inputting instance categories to outputting instance categories. The final results are also in the form of instance categories to ensure correct query closure. For example, the inner process of a query "Print the semi-complete callout symbols for '*Ra* 3.3' (without manufacturing methods, direction and machine allowance)" can be performed in the following order:

1. $\boldsymbol{Query_1}$: $\boldsymbol{OBJ_{Measurand}} \rightarrow \boldsymbol{Result_1}$

   (The selected arrows in $\boldsymbol{Result_1}$ is *Hom* = {*M#* → tolerance_type, *M#* → parameter_type, *M#* → parameter_name, *value* → parameter_extends}; The internal object (attribute) set in $\boldsymbol{Result_1}$ is *Att* = {*M#*, tolerance_type, parameter_type, parameter_extends | parameter_type = '***Ra***', parameter extends = *getParameterExtends*(3.3)})

2. $\boldsymbol{Query_2}$: $\boldsymbol{OBJ_{MeasurandToExtraction}} \rightarrow \boldsymbol{Result_2}$

   (The selected arrow set in $\boldsymbol{Result_2}$ is *Hom* = {***ME****#* → sampling_length, ***ME*** # → parameter type, ***ME*** # → parameter_extends, (parameter type × parameter_extends) → sampling_length}; The internal object (attribute) set in $\boldsymbol{TMP}_2$ is *Att* = {***ME****#*, sampling_length, parameter_type, parameter_extends | parameter_type, paramete_extends ∈ *Att* of $\boldsymbol{Result_1}$})

3. $\boldsymbol{Query_3}$: $\boldsymbol{OBJ_{Extraction}} \rightarrow \boldsymbol{Result_3}$

   (The selected arrow set in $\boldsymbol{Result_3}$ is *Hom* = {*E#* → sampling_length, *E#* → evaluation_length, *E#* → num_cutoff, (num_cutoff × sampling_length) → evaluation_length}; The internal object (attribute) set in $\boldsymbol{Result_3}$ is *Att* = {*E#*, sampling_length, evaluation_length, num_cutoff| sampling_length∈ *Att* of $\boldsymbol{Result_2}$})

4. $\boldsymbol{Query_4}$: $\boldsymbol{OBJ_{ExtractionAndMeasurandToFiltration}} \rightarrow \boldsymbol{Result_4}$

   (The selected arrow set in $\boldsymbol{Result_4}$ is *Hom* set = { *EF#*→ up_limit, *EF#*→ sampling_length, (sampling_length × parameter_type) → up_limit }; The inner object (attribute) set in $\boldsymbol{TMP}_4$ is *Att* = (*E#*, sampling_length, *F#*, up_limit, low_limit | up_limit ∈ *Att* of $\boldsymbol{Result_3}$ and parameter_type ∈ *Att* of $\boldsymbol{Result_1}$})

5. $\boldsymbol{Query_5}$: $\boldsymbol{OBJ_{Filtration}} \rightarrow \boldsymbol{Result_5}$

   (The selected arrow set in $\boldsymbol{Result_5}$ is *Hom* set = {*F#*→ up_limit, *F#*→ low_limit; up_limit→low_limit}; The internal object (attribute) set in $\boldsymbol{Result_5}$ is *Att* = {up_limit, low_limit, *F#* | up limit∈ *Att* of $\boldsymbol{Result_4}$})

6. $(\boldsymbol{OBJ}_{Measurand} \times \boldsymbol{OBJ}_{Extraction} \times \boldsymbol{OBJ}_{Filtration}) \rightarrow \boldsymbol{OBJ}_{Callout}$

   (The arrow set in $\boldsymbol{OBJ}_{Callout}$ is *Hom* set = { *C#* → num_cutoff, *C#* → up_limit, *C#* → low_limit, *C#* → tolerance_type, *C#* → parameter_type, *C#* → sampling_length, *C#* → evaluation _length, *C#* → value }; The internal object (attribute) set in P_4 is *Att* = { *C#*, num_cutoff, up_limit, low_limit, tolerance_type, parameter_type, sampling_length, evaluation_length, value | num_cutoff ∈ *Att* of $\boldsymbol{Result_3}$, up_limit ∈ *Att* of $\boldsymbol{Result_5}$, low_limit ∈ *Att* of $\boldsymbol{Result_5}$, tolerance_type ∈ *Att* of $\boldsymbol{Result_1}$, parameter_type ∈ *Att* of $\boldsymbol{Result_1}$, sampling_length∈ *Att* of $\boldsymbol{Result_3}$, evaluation_length ∈ *Att* of $\boldsymbol{Result_3}$, value ∈ *Att* of $\boldsymbol{Result_1}$})

The *M#*, *E#*, *F#* represent the unique identifiers of all instance categories created on the categories "*Measurand*", "*Extraction*", and "*Filtration*" respectively. The "*value*" is the transient internal object. The $\boldsymbol{Result_1}$, $\boldsymbol{Result_2}$, $\boldsymbol{Result_3}$, $\boldsymbol{Result_4}$, and $\boldsymbol{Result_5}$ represent sets of instance categories selected during the query process. For example, the $\boldsymbol{Result_1}$ represents instance categories for "*Measurand*" that was output from $\boldsymbol{Query_5}$. The arrows "*value* → parameter_extends", "parameter_type × parameter_extends) → sampling_length", "num_cutoff × sampling_length) → evaluation_length", "sampling_length → up_limit", and "up_limit→low_limit" are the corresponding method arrows. Thus, this query strategy enables the dynamic method queries during query processes. The "$\boldsymbol{OBJ}_{Measurand} \times \boldsymbol{OBJ}_{Extraction} \times \boldsymbol{OBJ}_{Filtration}$" is categorical product, which generates instance categories ($\boldsymbol{OBJ}_{Callout}$) for "Callout" with the selected arrrows.

This query strategy was implemented in the research based on the Simple Object Database Access (SODA) methods from DB4O with extensions of a set of Java methods to handle the functor mappings and compositions. In this example, the query is formed as the following Java clauses when the first time to create the "*Callout*" instance categories (See List 4.10).

```
query1.constrain(Measurand.class);
query1.descend("interObjId_measurand_paraType").descend("target").constrain("Ra").and(
query1.descend("interObjId_parameterExtends").descend("target").constrain((new
Measurand()).getParameterExtends(3.3)));
objectSet result1 = query1.execute();
result1.AddSelectArrows("interObjId_measurand_paraType");
result1.AddSelectArrows("interObjId_parameterExtends");
result1.AddSelectArrows("interObjId_tolerance_type");
Callout callout1 = result1.StoreSelectArrowsTo(Callout.class);
```

```
query2.constrain(MeasurandToExtraction.class);
query2.descend("interObjId_parameterExtends").descend("target").constrain(((Measurand)r
esult1.next()).getParaExtendArrow().getTarget()).and(query2.descend(("interObjId_measura
nd_paraType").descend("target").constrain((((Measurand)result1.next()).getParaTypeArrow
().getTarget())));
objectSet result2 = query2.execute();
result2.AddSelectArrows("interObjId_roughness_sampling_length");
result2.StoreSelectArrowsTo1(callout1);

query3.constrain(Extraction.class);
query3.descend("interObjId_parameterExtends").descend("target").constrain((new
Extraction()).getEvaluationLengthArrow(((MeasurandToExtraction)result2.next()).getSampli
ngLengthArrow().getTarget());
objectSet result3 = query3.execute();
result3.AddSelectArrows("interObjId_roughness_evaluation_length");
result3.AddSelectArrows("interObjId_num_cutOff");
result3.StoreSelectArrowsTo1(callout1);

query4.constrain(ExtractionAndMeasurandToFiltration.class);
query4.descend("interObjId_roughness_sampling_length").descend("target").constrain(((Me
asurandToExtraction)result2.next()).getSamplingLengthArrow().getTarget())and(query4.desc
end(("interObjId_measurand_paraType").descend("target").constrain((((Measurand)result1
.next()).getParaTypeArrow().getTarget())));;
objectSet result4 = query4.execute();
result4.AddSelectArrows("interObjId_up_limit");
result4.StoreSelectArrowsTo1(callout1);

query5.constrain(Filtration.class);
query5.descend("interObjId_low_limit").descend("target").constrain((new
Filtration()).getLowLimitArrow(((MeasurandToExtraction)result2.next()).getUpLimitArrow(
).getTarget());
objectSet result5 = query4.execute();
result5.AddSelectArrows("interObjId_Low_limit");
result5.StoreSelectArrowsTo1(callout1);
```

List 4.10: Java codes for a "*Callout*" query process.

The VirtualGPS system also offers SQL3 interface to form high-level SQL queries. This system can translate SQL queries into internal Java methods for retrieving information.

4.4.1.6 View Mechanism

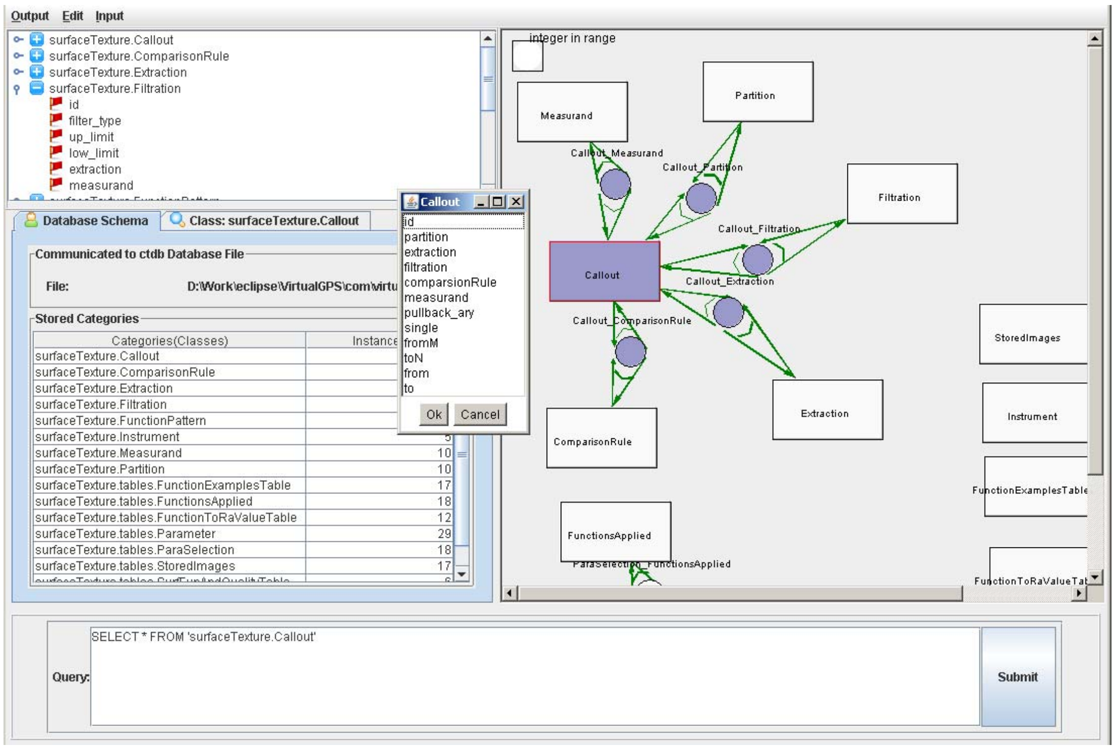
The view mechanism in the system was achieved through adding a set of selected arrows from a class category into its superclass category − "*Category*" − using the "*addArrows()*" method. Through defining unique view instructions, the categorical DBMS can offer customised views for different users.

4.4.1.7 Physical Storage Structures

Same as other relational and object-oriented DBMSs have both logic and internal schemas for databases, the categorical DBMS also contains a Category Theory based internal storage schema to store and retrieve objects physically. In this research, it was developed by extending the DB4O's physical storage mechanism. The DB4O's physical storage mechanism is based on the generics and reflection mechanisms in Java language. In order to keep the referential integrity, a special byte indication is automatically added to every instance category to indicate whether it is referred by other instance categories. If so, this instance category can not be deleted from the current DBMS operation.

4.4.2 The Visual Management Interface for Categorical DBMS

The categorical DBMS design also provided an embedded visual management interface for managing the stored GPS objects, their relationships, and constraints. It produces visual diagrams at runtime to represent all relevant data stored in the database according to the specified categorical object models. For example, as shown in Figure 4.5, by pressing the UI components on the visual diagram, the system can automatically generate optional query clauses for users and display the query results on screen.



Communicated to ctdb Database File

File: D:\Work\eclipse\VirtualGPS\com\virtu

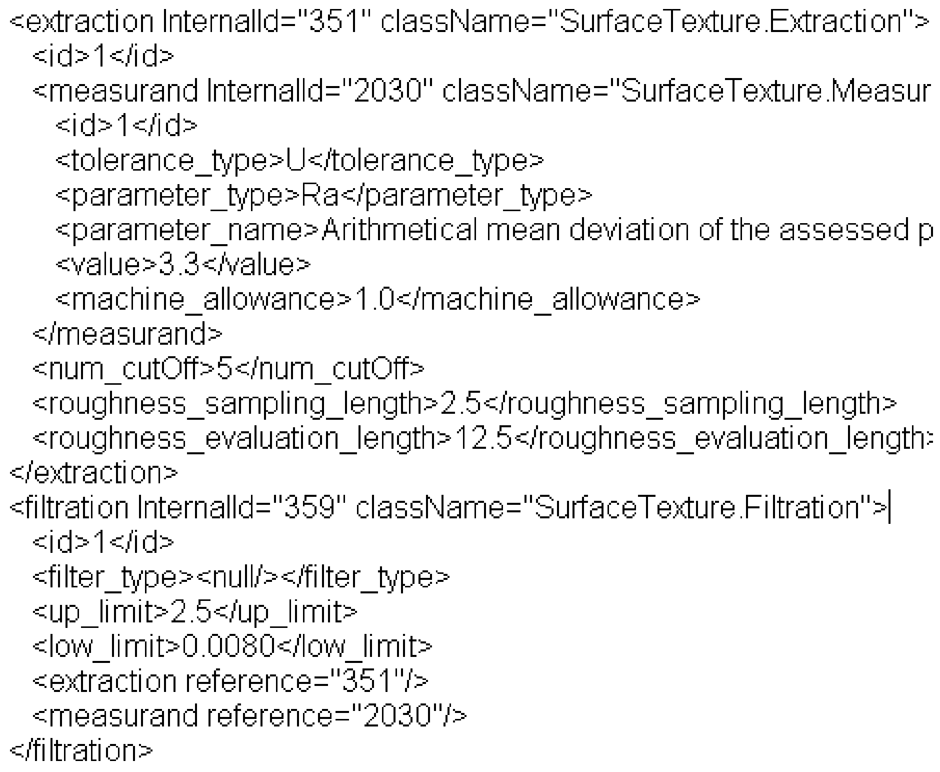
Stored Categories

Categories(Classes)	Instance
surfaceTexture.Callout	
surfaceTexture.ComparisonRule	
surfaceTexture.Extraction	
surfaceTexture.Filtration	
surfaceTexture.FunctionPattern	
surfaceTexture.Instrument	5
surfaceTexture.Measurand	10
surfaceTexture.Partition	10
surfaceTexture.tables.FunctionExamplesTable	17
surfaceTexture.tables.FunctionsApplied	18
surfaceTexture.tables.FunctionToRaValueTable	12
surfaceTexture.tables.Parameter	29
surfaceTexture.tables.ParaSelection	18
surfaceTexture.tables.StoredImages	17

Query:
```
SELECT * FROM 'surfaceTexture.Callout'
```

Figure 4.5: The main interface for the categorical DBMS.

This interface can also illustrate the metadata information (e.g. name and attribute types of a class category) for all class categories stored in the categorical DBMS. It can provide statistics on how many instance categories were created on a class category and the detailed information for a product relationship such as its cardinality, and participating class categories. Users can also update a class category (e.g. change types of attributes). After users presses the “Submit” button on the interface, a formed query will be executed with the query results being displayed on the relevant viewing windows. The Figure 4.6 shows the query results for a query clause “*SELECT * FROM ‘surfaceTexture Callout’*” in the XML format.



```
<extraction InternalId="351" className="SurfaceTexture.Extraction">
  <id>1</id>
  <measurand InternalId="2030" className="SurfaceTexture.Measur
    <id>1</id>
    <tolerance_type>U</tolerance_type>
    <parameter_type>Ra</parameter_type>
    <parameter_name>Arithmetical mean deviation of the assessed p
    <value>3.3</value>
    <machine_allowance>1.0</machine_allowance>
  </measurand>
  <num_cutOff>5</num_cutOff>
  <roughness_sampling_length>2.5</roughness_sampling_length>
  <roughness_evaluation_length>12.5</roughness_evaluation_length
</extraction>
<filtration InternalId="359" className="SurfaceTexture.Filtration">|
  <id>1</id>
  <filter_type><null/></filter_type>
  <up_limit>2.5</up_limit>
  <low_limit>0.0080</low_limit>
  <extraction reference="351"/>
  <measurand reference="2030"/>
</filtration>
```

Figure 4.6: An example for query results in XML format.

4.5 Summary

This chapter provided details on the design and implementations of a categorical DBMS that is a core part for realizing the VirtualGPS system. The categorical DBMS is a prototype to prove the applications of the Category Theory based object-oriented modelling. Although the current categorical DBMS implementation is not a full-fledged DBMS that can be compared with other commercial DBMSs, the research has demonstrated that the categorical DBMS can handle the complex operations such as storing and managing advanced data structures gained from current GPS standards with good consistency in database schema.

CHAPTER 5 DESIGN AND IMPLEMENTATION OF THE VIRTUALGPS SYSTEM

The Chapter 3 of the thesis has discussed how the Category Theory can be used as a mathematical foundation for the whole VirtualGPS system. This Chapter starts with a detailed introduction to the design of the VirtualGPS system, after which the implementation will be discussed. Finally, the Chapter concludes with a test case analysis to assess the design functions of the system.

5.1 The Design of the VirtualGPS System

The system design process conforms to the categorical incremental/refinement design process devised in Section 3.5 of Chapter 3. The following paragraphs give a complete example of the VirtualGPS system design.

5.1.1 The Categorical Business Map Construction for VirtualGPS

The first stage of the categorical design process is to design the categorical business map. In this design stage, the user requirements were captured. As described in the literature review of Chapter 2, the GPS matrix system is a universal tool for expressing geometrical requirements on product design drawings. It benefits product designers through providing a detailed description of functional requirements for geometrical products, and through reference to corresponding manufacturing and verification processes. Modern GPS standards aim at integrating all the data concerning essential steps of a production cycle right down to the macro or nano scale (ISO/TR 14638, 1995 [4]; ISO TC/213, 2001 [5]; Wang et al., 2004 [6]). The Figure 5.1 is the business scope of GPS, which links Function, Specification of macro- to nano-scale components, Manufacture, and Verification for different roles such as designers, production engineers and metrologists. Thus, they can exchange unambiguous information through the GPS specification.

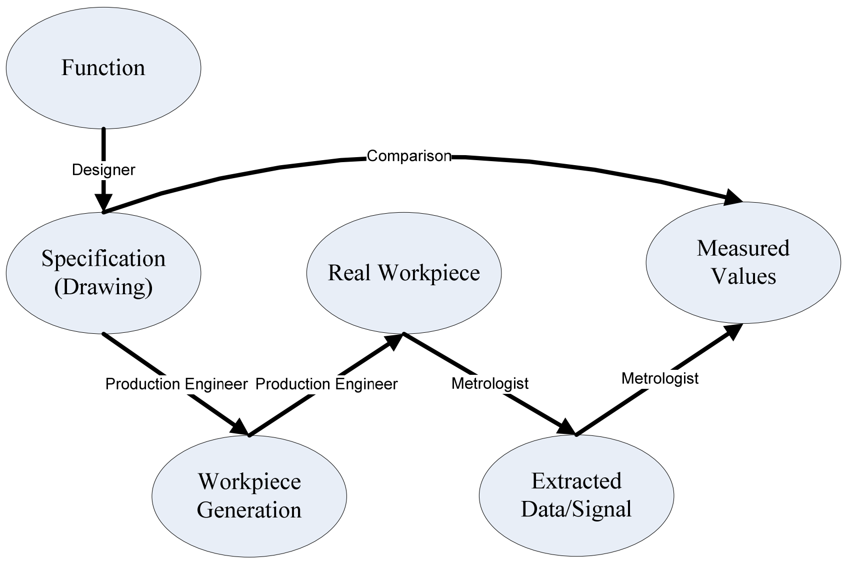


Figure 5.1: The business scope of GPS.

The Figure 5.1 shows how GPS standards can be related to a complete industrial procedure of producing a geometrical workpiece: design of the workpiece by setting up unambiguous specifications, manufacture of the workpiece under the guidance of specifications, and metrology of workpiece through the verification of specification. This also illustrates that the verification is of vital importance for modern manufacturing industries, since verification results can be evaluated to refine the GPS standards relating to new GPS parameters, suggest specific tolerance values and update manufacturing procedures and so on. Therefore, a cyclic quality chain for refining the quality of geometrical products can be formed. The Figure 5.2 gives an example of this quality chain on a surface manufacture.

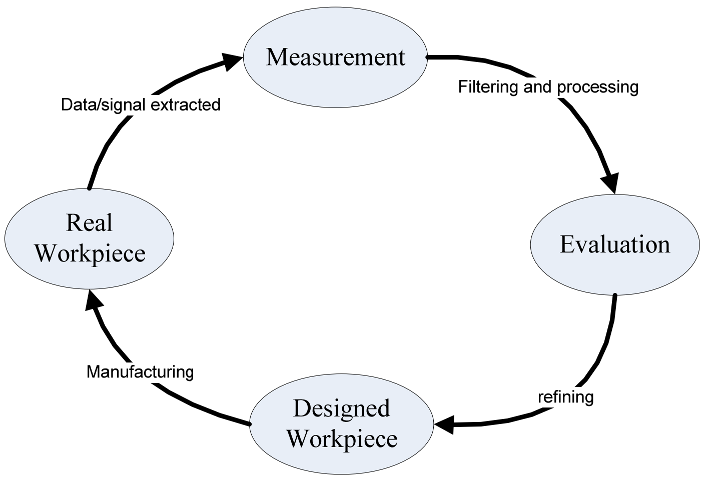


Figure 5.2: A cyclic surface quality chain.

The "*Measurement*" and "*Evaluation*" in Figure 5.2 are both relating to "metrologist" in Figure 5.1.

Furthermore, a high level categorical business map for the VirtualGPS system can be formed as Figure 5.3 demonstrated.

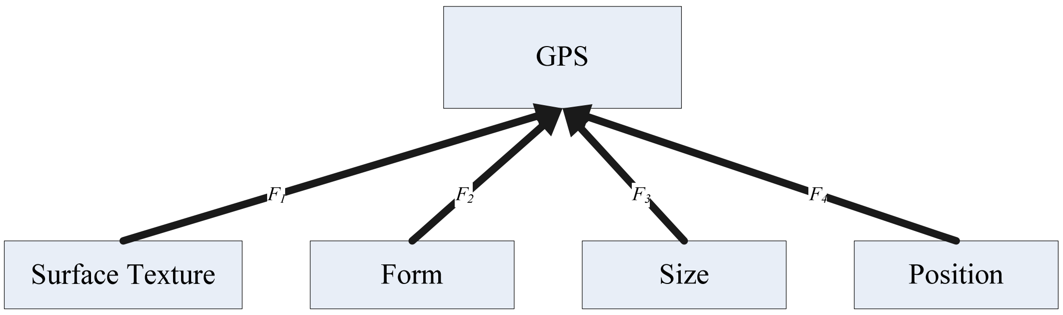


Figure 5.3: Overall framework of the VirtualGPS system.

Figure 5.3 shows that the proposed VirtualGPS framework contains four main knowledge bases (Surface Texture, Form, Position and Size), describing different knowledge domains in a categorical view — each knowledge base becomes a module (category) and faithful functors are used to inject these four modules (categories) into the VirtualGPS system.

5.1.2 The Categorical Analysis Model Construction for VirtualGPS

The second stage of the categorical design process is to design the categorical analysis model. This project has so far partially completed the Surface Texture and the Form modules. Therefore, this thesis focuses on illustrating the design and implementation details of the Surface Texture module, which can be further divided into four sub-knowledge bases based on Figures 5.1 and 5.2: Function, Specification, Manufacture, and Verification. These four sub-knowledge bases become components of the Surface Texture module.

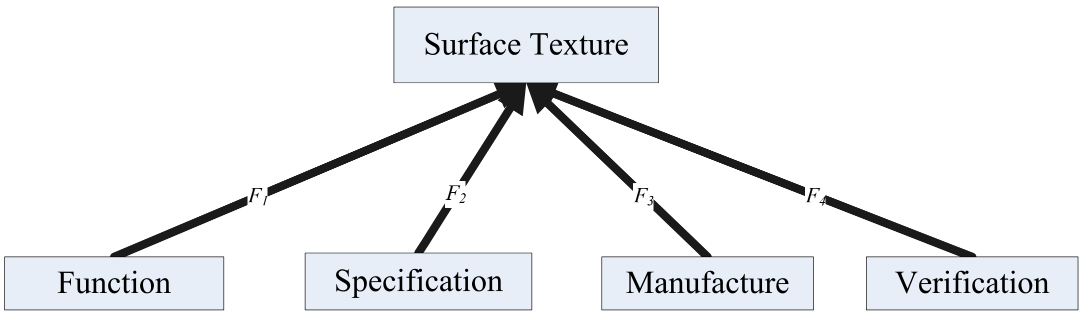


Figure 5.4: Components in Surface Texture module.

In Figure 5.4, the four components of Surface Texture module are represented as lower level categories.

5.1.3 The Categorical Design Model Construction for VirtualGPS

The third stage of the categorical design process is to design the categorical design model. The categorical design model is used to detail and refine the four components: Function, Specification, Manufacture and Verification. The detailed designs of these four components are discussed in Section 5.2, 5.3, and 5.4, which use same process as discussed here to refine each component.

5.1.4 The Categorical Sequence Diagram Construction for VirtualGPS

The fourth stage of the categorical design process is to create the sequence diagrams for modules in VirtualGPS. Figure 5.5 shows a sequence diagram for the Surface Texture Module.

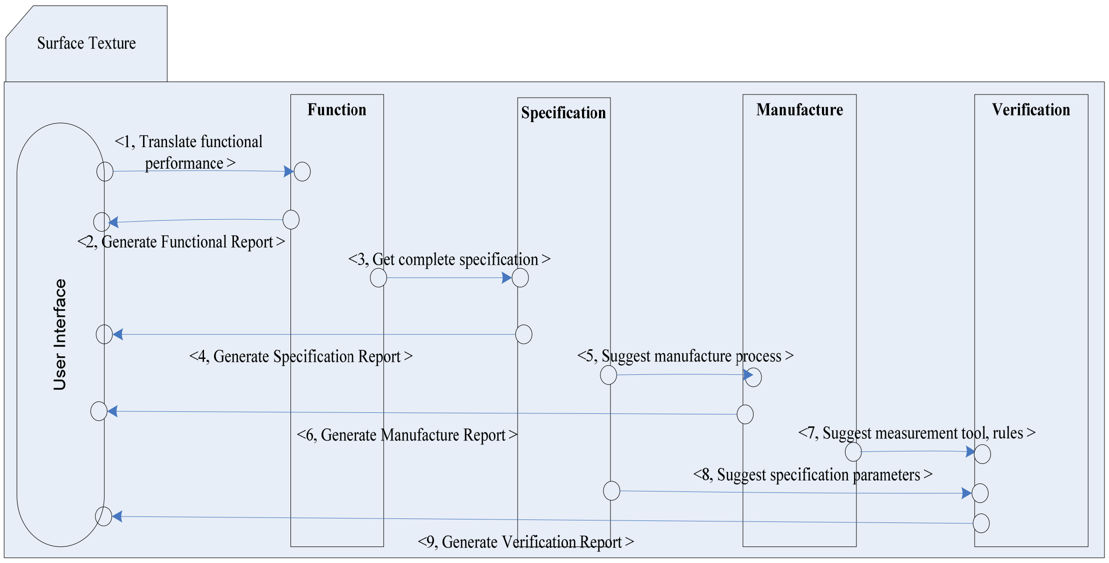

Surface Texture
Function
Specification
Manufacture
Verification
User Interface
<1, Translate functional performance >
<2, Generate Functional Report >
<3, Get complete specification >
<4, Generate Specification Report >
<5, Suggest manufacture process >
<6, Generate Manufacture Report >
<7, Suggest measurement tool, rules >
<8, Suggest specification parameters >
<9, Generate Verification Report >

Figure 5.5: The sequence diagram for the Surface Texture module.

The diagram highlights the perceived process flow for utilising the Surface Texture module in a typical manufacturing cycle, which can be described as follows:

(**For designers**) Product designers activate the VirtualGPS system; the "Function" component will search and advise users by translating functional performances (e.g. fluid friction or dry friction) into surface texture parameters defined in GPS-matrices; and then generates a function analysis report using a so-called "pattern" language. Therefore, the function component is responsible for translating the design intent into requirements of GPS characteristics for designers.

(**For designers and manufacturing engineers**) The generated "Specification" component produces the details of the GPS specification on the technical drawing in the form of complete 'callouts', based on the selected surface texture parameters.

(**For manufacturing engineers**) In accordance with the deduced specification report and any extra criteria defined (such as material types and quantity), the "Manufacture" component can suggest appropriate manufacturing processes for the designers. In order to enable cross comparing among different processes, a manufacturing process report for each recommended process plan will be formed, which includes details such as process description, material suitability, process variations, costing issues and sample applications.

(**For metrologists**) The final "Verification" component enables metrologists to choose from recommended measurement instruments and filtering techniques to formulate a measurement strategy.

5.1.5 The Categorical Deployment Model Construction for VirtualGPS

The fifth stage of the categorical design process is to divide the design classes of the design model into subsystems based on outputs of Section 5.1.4.

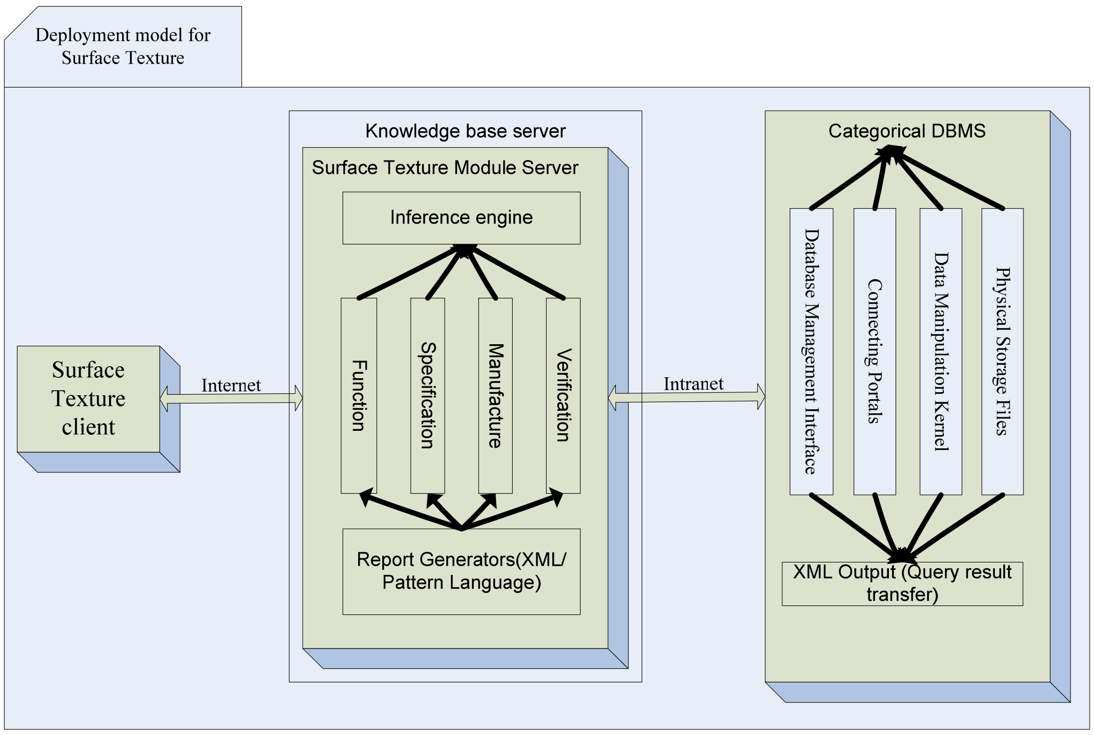


Figure 5.6: Overall architecture of the VirtualGPS system.

The overall system architecture for the devised VirtualGPS is illustrated in Figure 5.6. The system design has adopted a classic three-tier architecture (the accessing client, the knowledge manipulation server, and the database server). Object-oriented concepts and techniques have been adopted to ensure encapsulation and system robustness with several rigidly defined interfaces for inter-module operations. With the modularized design and its inherent structural adaptability, this system can change existing features and functions or add new ones efficiently.

The client-side browser provides users with an interface to access GPS knowledge organised by rules and standards devised in the knowledge base. Moreover, users can also add new information using the pattern language format offered by the user interfaces of the VirtualGPS system. This system can then automatically organize the inputted information into knowledge for users. Each knowledge base (module) of the knowledge-based system contains four components: Function, Manufacture, Specification, and Verification as explained earlier sections. The former two can generate and export reports for reference in a pattern language style format, while the latter pair can output reports in the XML format for web-based operations. The system supports web applications through secured socket communications for knowledge distribution and sharing on Intranet and Internet.

Also shown in Figure 5.6, for forming accurate and comprehensible 'knowledge' from the maze of GPS-matrices, this project had also developed a back-end database and its management system based on Category Theory to store the complex GPS-matrices and their constraints. The database is referred as the "categorical DBMS" in this thesis based on its nature of adopting Category Theory notions for forming the database model.

Based on the explanations of the previous four sub-Sections (from Section 5.1.1 to Section 5.1.4), the overall architecture of the VirtualGPS has been defined. This high level architecture can clarify the following software aspects:

- Which modules should be contained in this system.
- Which components should be included in each module.
- What are the computing functions of these modules and components.
- How these modules and components interact or communicate with each other.
- How these modules or components are deployed on the computing resources.

After getting this high level architecture of the VirtualGPS system, several lower level refinement and incremental processes need to be taken to get details for design classes contained by each component. Therefore, components are also designed following the five stages of the categorical design process, and the outputs of component designs should clarify the following aspects:

- Which design classes are contained in each component.
- What are the computing functions of these design classes.

- How the design classes interact or communicate in a component.

The Sections 5.2, 5.3, 5.4 and 5.5 give detailed explanations on designs of the Function, Specification, Manufacture and Verification components in the Surface Texture module. These four components in other modules such as Form, Size and Position can be designed in the same manner after acquiring enough knowledge.

5.2 The Function Component Design

This section aims to provide a detailed discussion of the design of the Function component, which focuses on discussing the knowledge acquisition and knowledge organization. The Function component is used to suggest surface roughness parameters based on functional performance analysis.

5.2.1 The Categorical Business Map Model Construction for Function

This step is used to analyze the user requirements and then define the business map to illustrate the business scope for the Function component in a high abstraction level. Surfaces can be divided into two types: functional and non-functional (Mummery, 1990 [91]). A non-functional surface means the surface does not affect the quality of the product, which can be either mirror smooth or sand paper roughness. However, a functional surface has a function that is closely related to the quality of the product. For example, the outside of an engine block is a non-functional surface ,which has no specific function; while the contact area between the cylinder liner and the piston rings are functional surfaces performing the sealing function. For the functional surface, the surface texture has a direct influence on the quality of the product. Therefore, quality of a geometrical product can be optimized by analysing the relationship between surface topography and function of the product (Mummery, 1990 [91]).

However, because of the lack of references, it is difficult for users to select appropriate surface texture parameters and their corresponding tolerance values specified on technical drawings. Thus, the Function component in the VirtualGPS system aims to help users to select surface texture parameters according to surface functions. The main software functions for the Function component can be summarized as:

- Provide suitable surface texture parameters with tolerance values based on exiting cases.
- Provide a set of parameters selection rules for user references. Users can select

suitable parameters through systematic consideration of these selection rules.

- Provide an open platform for users to add their expert knowledge on specific cases.

Based on these user requirement captures, a business map can be built, see Figure 5.7.

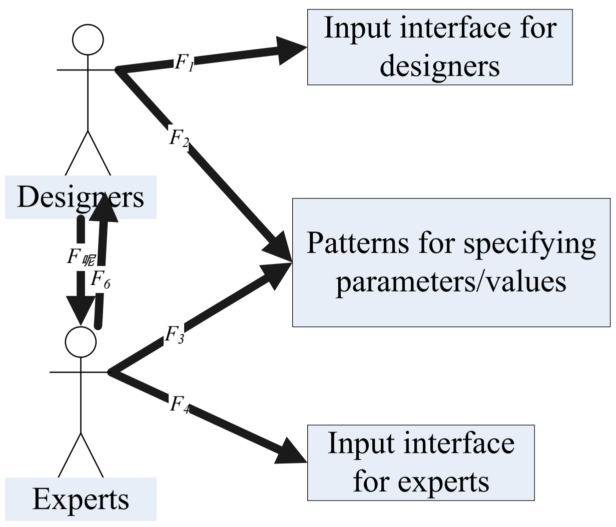

Input interface for designers
F_1
F_2
Designers
Patterns for specifying parameters/values
F_5
F_6
F_3
F_4
Input interface for experts
Experts

Figure 5.7: The business map for Function component.

Figure 5.7 demonstrates that two users of differing expertise can interact with the Function component:

(1) **Designers**. Designers input surface requirements, functions or surface tribology and then the system infers suitable roughness parameters with tolerance values. The parameters and tolerance values should be organized in a specific pattern language format.

(2) **Experts**. Experts can add new knowledge by using the same pattern language format to form new cases.

The roles for designers and experts can be interchanged. For example, designers can add new knowledge based on their design experiences. By doing like this, the quality of products can be improved constantly.

5.2.2 The Categorical Analysis Model Construction for Function

Based on the discussions in Section 5.2.1, the core target for Function component is finding the relationship between functions and surface texture parameters. However, at present, there is no a systematic way to define all aspects of functions and to link them to surface texture parameters. At the moment, only a little information can be retrieved from GPS to link surface texture parameters with functions. Therefore, at present, the Function component can not totally support reasoning about surface texture parameters and tolerance values based on the arbitrary functions or surface descriptions inputted by users. However, the Function component provides a basis of a guidance procedure for linking the functions with surface texture parameters, which was achieved by using some basic inference rules retrieved from GPS. Users can use them to find the suitable surface texture parameters for their specific tasks, and then store these tasks as cases in the VirtualGPS system in a specific pattern format. As the number of cases increases and relating GPS standards are enriched, the Function component will be trained, and then the well-trained Function component can make use of fuzzy logic to infer surface texture parameters with suitable tolerance values that match the functions. The Category Theory based pattern language for organizing the cases has been discussed in Section 3.7.1 of Chapter 3, so this section will only discuss the guidance procedure for choosing surface texture parameters with several inference rules. The procedure can be simplified as following:

(1) **Determine surface requirements according to surfaces**. This can be achieved by investigating the counter parts, the properties and the relative motion of the workpiece, and then specify the surface requirements to match those attributes.

(2) **Determine the classification of Functions**. The classification of functions is very difficult because they are so numerous and diverse that is impossible to carry out a systematic approach to cover them. Actually, at present, tribology is a good tool for function classification. Tribology is the science and technology of friction, wear and lubrication (Kalpakjian and Schmid, 2005 [92]). The Figure 5.8 is classification of functions by using tribological applications such as contact, wear, lubrication and failure mechanism, which is a simplistic generic approach to provide basic guidance (Whitehouse, 2002 [93]).

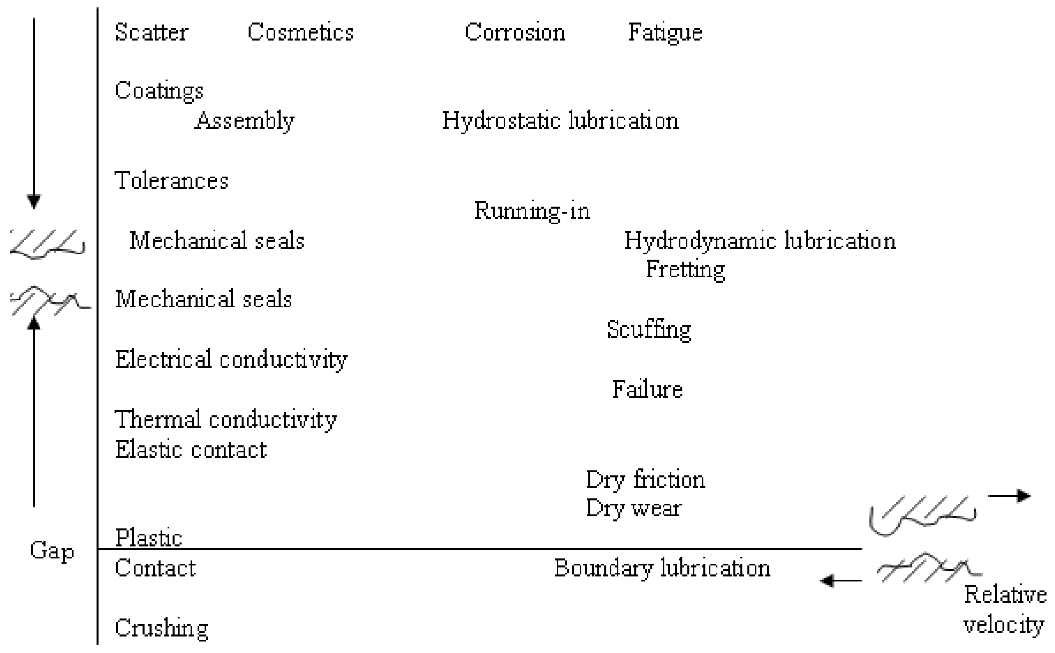
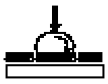
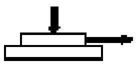
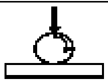
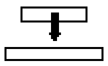


Figure 5.8: The function map.

The horizontal axis of Figure 5.8 represents the relative velocity of the two surfaces, while the vertical axis represents the gap between the two surfaces. The scales are omitted from the diagram. It is supposed that the vertical axis is in micrometers and the horizontal axis has a maximum realistic value of 5m/sec.

(3) **Match different functions with different surface requirements**. This can be achieved by finding a relationship between tribology and surface requirements. Once the relationship between tribology and surface requirements are defined, the transitive relationships between functions and surface requirements can be defined. The Table 5.1 is another function classification by using types of wears, relative motions and contact bodies (Filetin, 2002 [94]).

Elements	Relative motions		Type of wear	Mechanism of wear			
	Type	Schemes	Type	AD	AB	WF	TC
Solid body Lubricant Solid body	Sliding		Hydrodynamic			●	○
Solid body Solid body	Sliding		Sliding wear	●	○	○	●
	Rolling		Rolling wear	○	○	●	○
	Impact		Impact wear	○	○	●	○

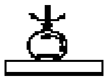
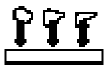
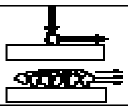
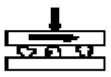
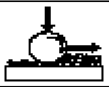
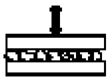
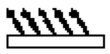
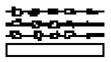
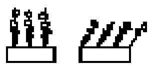
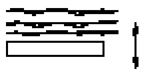
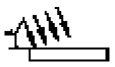
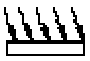
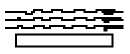
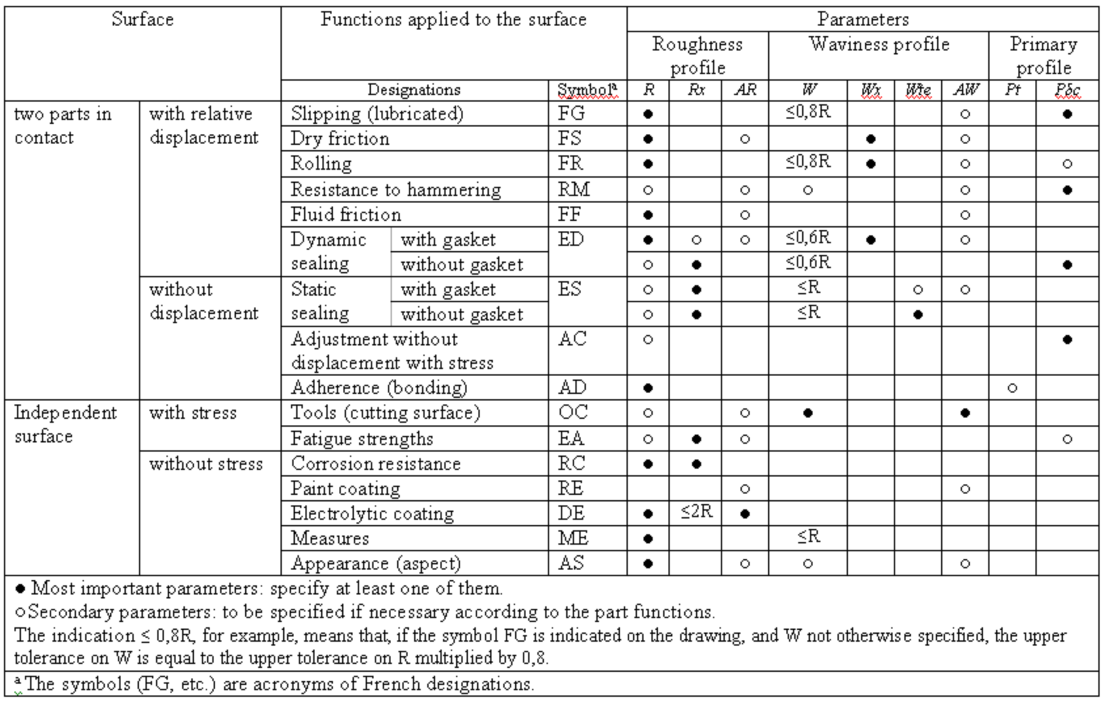
	Vibration		Fretting	●	●	●	●
Solid body Particles	Impact		Abrasion		●	●	○
	Sliding		Abrasion		●		○
Solid body Particles Solid body	Sliding		Abrasion	○	●	●	○
	Rolling		Abrasion	○	●	●	○
	Impact		Abrasion	○	○	●	○
Solid body Particles Fluid	Flow		Erosion		●	●	○
Solid body Particles Gas	Flow		Erosion	○	●	●	○
			Erosion	○	●	●	○
Solid body Fluid	Flow		Cavitation Erosion			●	○
	Impact		Erosion			●	○
	Flow		Erosion by fluids			○	●
Solid body Gas	Gas Erosion		Cavitation Erosion				●
AD – Adhesion, AB – Abrasion, WF – Wear Fatigue, TC – Tribocorrosion, ●Most important ○ Less important							

Table 5.1: Examples of functions.

(4) **Select surface parameters with corresponding tolerance values for surface requirements**. By doing this, the relationship between surface texture parameters and functions can be transitively defined. Therefore, inferences from functions to surface parameters can be achieved.

Besides this guidance procedure, several other tables gathered from GPS matrices are also included in the VirtualGPS. For example, Table 5.2 shows relationships between motif parameters and functions of surface, defined in ISO 12085 1996 (ISO 12085, 1996 [25]).



Surface		Functions applied to the surface			Parameters								
					Roughness profile			Waviness profile				Primary profile	
		Designations		Symbol[a]	*R*	*Rx*	*AR*	*W*	*Wx*	*Wte*	*AW*	*Pt*	*Pδc*
two parts in contact	with relative displacement	Slipping (lubricated)		FG	●			≤0,8R			○		●
		Dry friction		FS	●		○		●		○		
		Rolling		FR	●			≤0,8R	●		○		○
		Resistance to hammering		RM	○		○	○			○		●
		Fluid friction		FF	●		○				○		
		Dynamic sealing	with gasket	ED	●	○	○	≤0,6R	●		○		
			without gasket		○	●		≤0,6R					●
	without displacement	Static sealing	with gasket	ES	○	●		≤R		○	○		
			without gasket		○	●		≤R		●			
		Adjustment without displacement with stress		AC	○								●
		Adherence (bonding)		AD	●							○	
Independent surface	with stress	Tools (cutting surface)		OC	○		○	●			●		
		Fatigue strengths		EA	○	●	○						○
	without stress	Corrosion resistance		RC	●	●							
		Paint coating		RE			○				○		
		Electrolytic coating		DE	●	≤2R	●						
		Measures		ME	●			≤R					
		Appearance (aspect)		AS	●		○	○			○		

● Most important parameters: specify at least one of them.
○ Secondary parameters: to be specified if necessary according to the part functions.
The indication ≤ 0,8R, for example, means that, if the symbol FG is indicated on the drawing, and W not otherwise specified, the upper tolerance on W is equal to the upper tolerance on R multiplied by 0,8.

[a] The symbols (FG, etc.) are acronyms of French designations.

Table 5.2: Parameters selection example.

All these tables for guiding the selection of surface texture parameters from functions become class categories that in turn serve as inference rules for the Function component.

5.2.3 The Categorical Design Model Construction for Function

In the previous section, a set of class categories for inferring surface texture parameters have been determined. At the design model construction stage for the Function component, these class categories are refined by defining their lower level subclass categories, which themselves form tree structures, to represent these tables specified in Section 5.2.2. This section takes Table 5.2 as an example. The Figure 5.9 illustrates a tree structure that is represented in the categorical way.

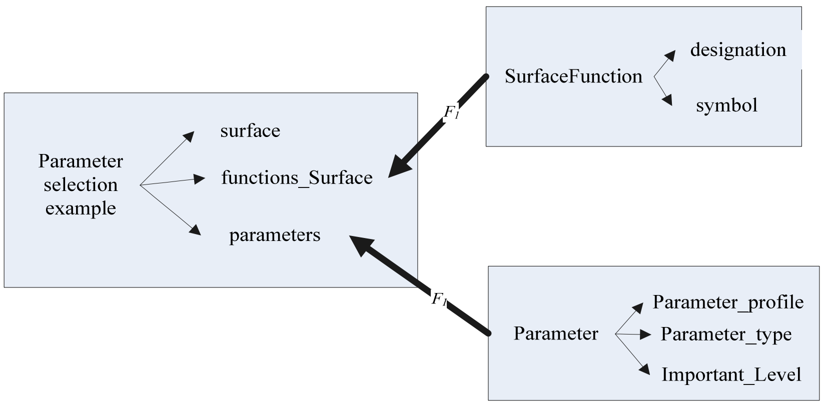


Figure 5.9: Categorical representation of parameters selection example table.

Figure 5.9 shows that the complex tree structure of Table 5.2 can be gracefully modelled, and which is difficult for relational data model to handle it. After defining design classes (class categories) for Function component, the next stage is specifying the interactions among these class categories belonging to the Function component.

5.2.4 The Categorical Sequence Diagram Construction for Function

The design classes discussed in Section 5.2.3 serve as inference rules in the Function component, which can be grouped into three sub-components: inference rules for linking surface requirements with surface texture parameters; inference rules for linking functions with surface texture parameters; and inference rules used in cases.

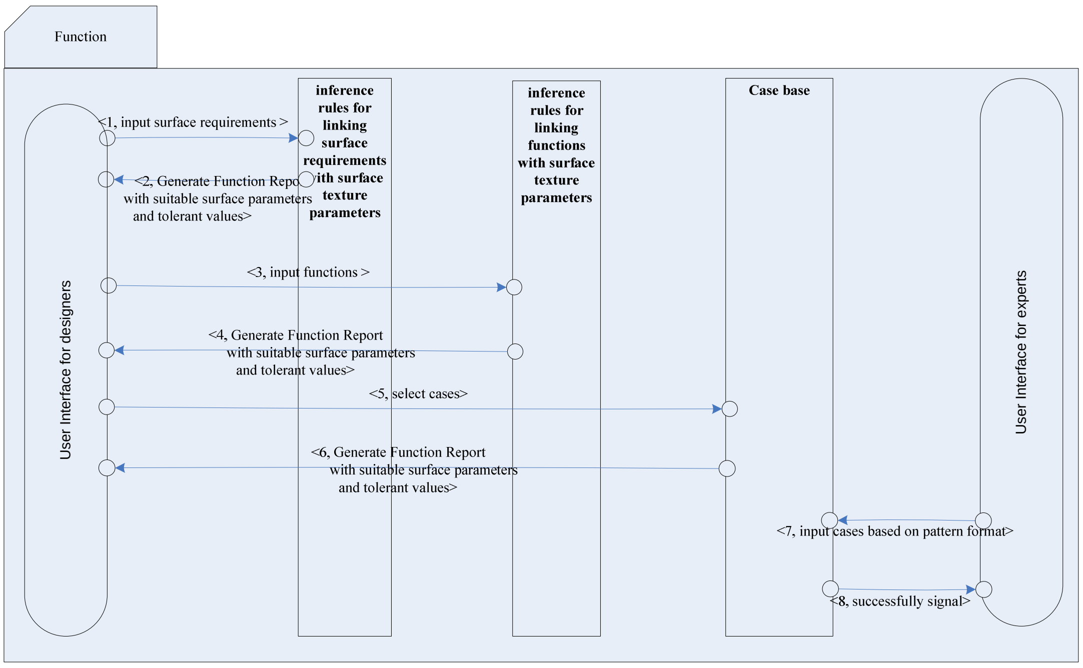
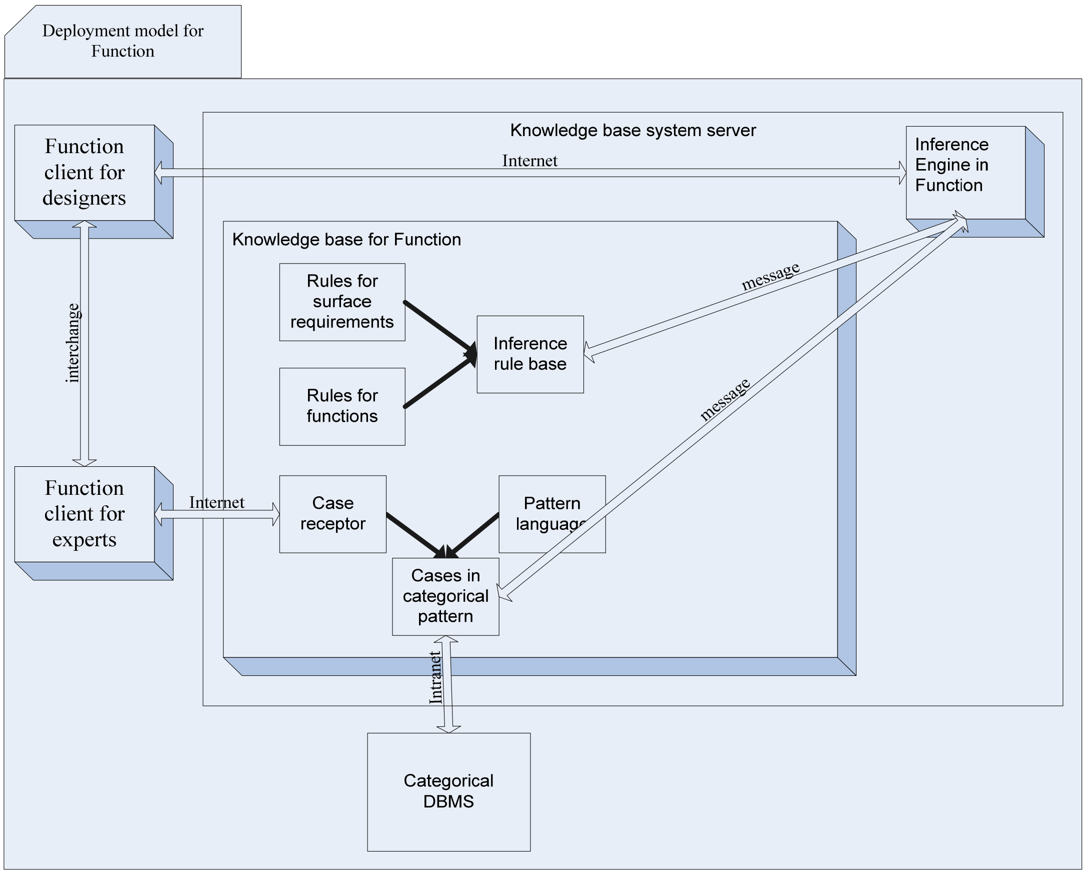

Function
User Interface for designers
inference rules for linking surface requirements with surface texture parameters
inference rules for linking functions with surface texture parameters
Case base
User Interface for experts
<1, input surface requirements>
<2, Generate Function Report with suitable surface parameters and tolerant values>
<3, input functions>
<4, Generate Function Report with suitable surface parameters and tolerant values>
<5, select cases>
<6, Generate Function Report with suitable surface parameters and tolerant values>
<7, input cases based on pattern format>
<8, successfully signal>

Figure 5.10: The sequence diagram for Function component.

Figure 5.10 is organized according to business requirements defined in Figure 5.7. However, as stated earlier, the current VirtualGPS can not support inferences of surface texture parameters according to descriptions or key words for specifying surface requirements or functions due to lack of knowledge in GPS. The prototype VirtualGPS can only suggest surface parameters with corresponding tolerance values based on existing case scenarios. Thus, we call this system as knowledge-based system or partially rule-based expert system. However, when cases and other related knowledge are enriched, the VirtualGPS will become a real rule-based expert system.

5.2.5 The Categorical Deployment Model Construction for Function

Based on analysis of the sequence diagram that shows the interactions between design classes in the Function component, a deployment topological graph for specifying the allocation of these design classes on computing resources can be constructed as Figure 5.11 demonstrates.



Deployment model for Function
Function client for designers
Knowledge base system server
Internet
Inference Engine in Function
Knowledge base for Function
Rules for surface requirements
message
interchange
Inference rule base
Rules for functions
message
Function client for experts
Internet
Case receptor
Pattern language
Cases in categorical pattern
Intranet
Categorical DBMS

Figure 5.11: The deployment topological graph for Function component.

The inference rule base in Figure 5.11 contains two parts: rules for surface requirements and rules for functions. A product relationship is built between case receptor and class categories for organizing the pattern language to form instances of cases in categorical pattern language format. These case instances will be stored in categorical DBMS for future case based inferences.

5.3 The Specification Component Design

This section aims to provide a detailed discussion of the Specification component's design, which focuses on discussing the knowledge acquisition and organization. The Specification component is used to provide detailed geometrical specifications for the selected surface parameters, including information obtained from partition, extraction, filtration and comparison processes. The main software functions for the Specification component can be summarized as:

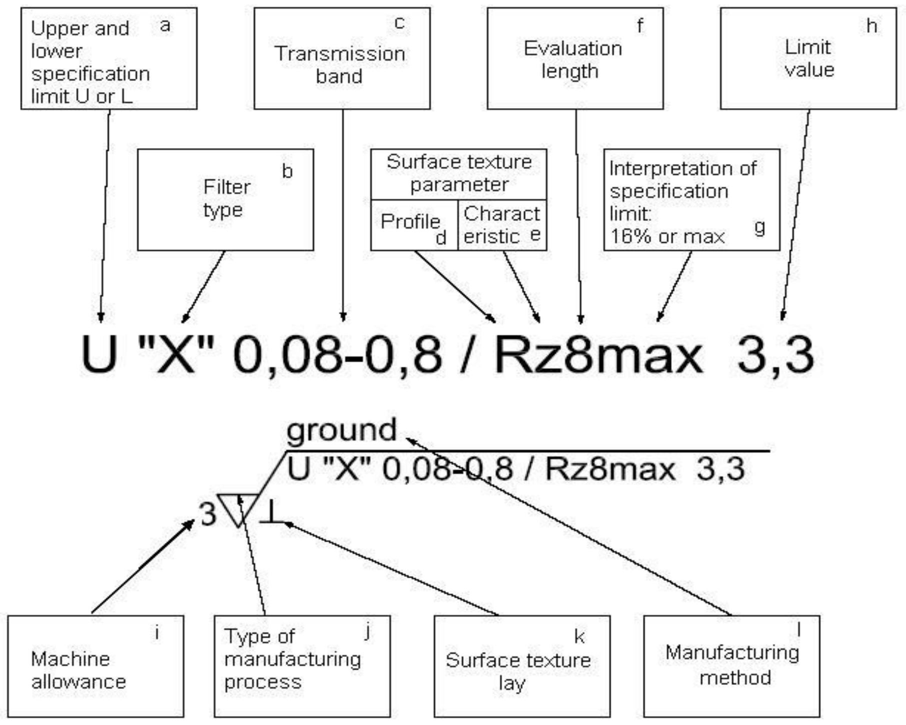
- Generate a complete callout from a simple callout on a drawing by providing default values. For example, the complete callout "U 0.008-2.5 / Ra516% 3.3" can be generated for the simple callout "*Ra* 3.3".
- Explain each symbol in a complete callout in detail. Users can get detail explanations and descriptions of each symbol in a complete callout. For example, parameter names will be rendered for each parameter type.
- Referred to as basis for Manufacture and Verification components. This includes two aspects:

  (1) The complete callout is "shallow" knowledge for inferring "depth" knowledge in Manufacture and Verification.

  (2) The complete callout can guide operations in Manufacture and Verification. For example, the comparison rule in a complete callout is used in a comparison process of Verification. The methods for the "max-rule" or "16%-rule" will be programmed in the Verification component according to the indications of complete callouts.

5.3.1 The Categorical Business Map Construction for Specification

After capturing the user requirements, the inputs and the outputs of Specification component have been determined: the inputs of the Specification component are specific surface parameters while the outputs of the Specification component are complete "callout" symbols gathered from three feature operations: partition, extraction and filtration. The Figure 5.12 shows the structure for a complete callout symbol (ISO 1302, 2002 [95]).



Upper and lower specification limit U or L a
Transmission band c
Evaluation length f
Limit value h
Filter type b
Surface texture parameter
Profile d
Charact eristic e
Interpretation of specification limit: 16% or max g
U "X" 0,08-0,8 / Rz8max 3,3
ground
U "X" 0,08-0,8 / Rz8max 3,3
3
Machine allowance i
Type of manufacturing process j
Surface texture lay k
Manufacturing method l

Figure 5.12: Complete surface texture callout symbol.

The key explanations for Figure 5.12 are:

a. Indication of specification limit.
b. Filter type "X".
c. The transmission band, including the lower limit and the upper limit.
d. Profile (*R* – roughness profile, *W* – waviness profile or *P* – primary profile).
e. Characteristic/parameter.
f. Evaluation length as the number of sampling lengths.
g. Comparison rule ("16 %-rule" or "max-rule").
h. Limit value in micrometres.
i. Machining allowance.
j. Type of manufacturing process.
k. Surface texture lay.
l. Manufacturing methods.

The measurement of the surface texture is generally determined in terms of its roughness, waviness and form. The roughness is the process marks or witness marks produced by the action of the cutting tools or machining processes, but may include other factors such as the structure of the material. The waviness is usually produced by instabilities in the machining process, such as an imbalance in a grinding wheel, or by deliberate actions in the machining process. Waviness has a longer wavelength than roughness which is superimposed on the waviness. The form is the general shape of the surface, ignoring variations due to roughness and waviness. Deviations from the desired form can be caused by many factors. There are three principal groups of surface texture parameters relating to this project: profile parameters defined in ISO 4287 (e.g. amplitude parameter *Ra*, spacing parameter *Rsm*, and hybrid parameter *Rda*) (ISO 4287, 1997 [96]), motif parameters defined in ISO 12085 (e.g. mean motif height *R*, mean motif width *AR* and maximum motif height *Rx*) (ISO 12085, 1996 [25]) and parameters based on material ratio curve defined in ISO 13565-2 and ISO 13565-3 (e.g. *Rk*, *Rpk*, *Rvk*, etc.) (ISO 13565-2, 1996 [97]; ISO 13565-3, 2000 [98]). Detailed introductions on these surface texture parameters can also be found in the book − "Geometrical Product Specifications Course for Technical Universities", which will not be described in this thesis (Humienny et al., 2001 [3]). Figure 5.12 illustrates that besides the surface texture parameters and their corresponding limit values (tolerance values), there is also some other information relating to the partition, extraction, filtration and comparison operations. For example, the complete callout for "Ra 3.3" is "0.008-2.5/Ra516% 3.3", where the missing information is supplied by the ISO 1302 (ISO 1302, 2002 [95]). By organizing the surface texture parameters, tolerance values, and other information relating to partition, extraction, filtration and comparison operations into complete callouts, the knowledge base for the Specification component can be formed. Therefore, the Specification component can relieve users from the burden of cross referencing a set of ISO file based papers to obtain the complete GPS specifications.

Based on the user requirement analysis above, the knowledge base of the Specification component has to contain five use cases: the measured surface partition, finite data point extraction, profile filtration, measurand definition, and the chosen comparison rule. Therefore, the business map for Specification component was built as Figure 5.13.

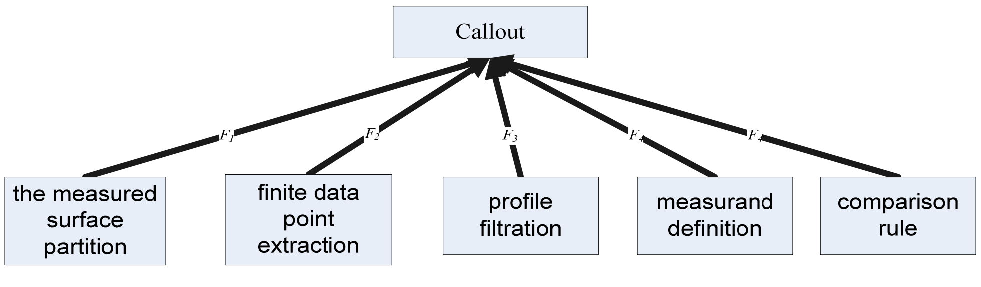


Figure 5.13: The business map for Specification component.

5.3.2 The Categorical Analysis Model Construction for Specification

After getting the business map for Specification component as Figure 5.13 illustrated, the next step is to analyse the computing functions of all these use cases defined in business map and then obtain a set of analysis classes. According to ISO files for GPS, the following use case refinements can be clarified:

(1) **Measurand definition**. In the Specification component under the Surface Texture module, the measurand is the surface texture parameters defined in the GPS. The surface texture parameters in the GPS contain several types of affiliating information: tolerance type, parameter type, parameter name, tolerance value, machining allowance. The parameter type includes two parts: profile indication and characteristic indication, where the profile has three possible indications: *R* (roughness profile), *W* (waviness profile) and *P* (primary profile). For example, in *Rz*, the "*R*" indicates the roughness profile and "*z*" indicates the characteristic feature. There are three groups of surface texture parameters in GPS to deal with these three kinds of profile. The Table 5.3 shows a set of surface texture parameters contained in material ratio curve group (ISO 13565-2, 1996 [97]; ISO 13565-3, 2000 [98]).

	Parameters				
R-profile parameters based on linear material ratio curve	*Rk*	*Rpk*	*Rvk*	*Mr1*	*Mr2*
	Rke	*Rpke*	*Rvke*	*Mr1e*	*Mr2e*
R-profile parameters based on the material probability cure	*Rpq*	*Rvq*		*Rmq*	
p-profile parameters based on the material probability curve	*Ppq*	*Pvq*		*Pmq*	

Table 5.3: Parameters based on material ratio curve.

Therefore, an analysis class (class category) "*Measurand*" can be defined with internal objects: tolerance_type, parameter_type, parameter_name, parameter_category, tolerance value and machine_allowance.

(2) **Surface Partition**. The feature operation − partition is used to identify the bounded surface which is to be characterized. The bounded surface texture is influenced by the detailed form of the profile curve, while the profile curve is usually determined by the manufacturing processes (ISO 1302, 2002 [95]). Therefore, the feature information for partition includes the direction of surface texture lay, the manufacture type and manufacture methods of the

surface, which catch the initial properties of the surface being evaluated. The detailed introduction of the feature information relating to partition can be referred in ISO 1302 (ISO 1302, 2002 [95]). Therefore, an analysis class (class category) "*Partition*" can be defined with internal objects: direction_symbol, direction_definition, manufacture_type_symbol, manufacture_type_meaning, and manufacture_method.

(3) **Finite data point extraction**. The feature operation extraction is used to determine a finite number of points on the surface that are extracted for measurement and evaluation. The feature information for extraction includes number of sampling lengths (num_cutOff) and evaluation length of the evaluated surface. The num_cutOff indicates the number of sampling lengths within an evaluation length. A cut_off is the wavelength which is used as a means of separating or filtering the wavelengths of a surface. Sampling length is the length in the direction of the X-axis used for identifying the irregularities characterizing the profile under evaluation (ISO 4287, 1997 [96]). The Figure 5.14 shows an example of the relationships of traverse length, evaluation length, and sampling length (cut_off).

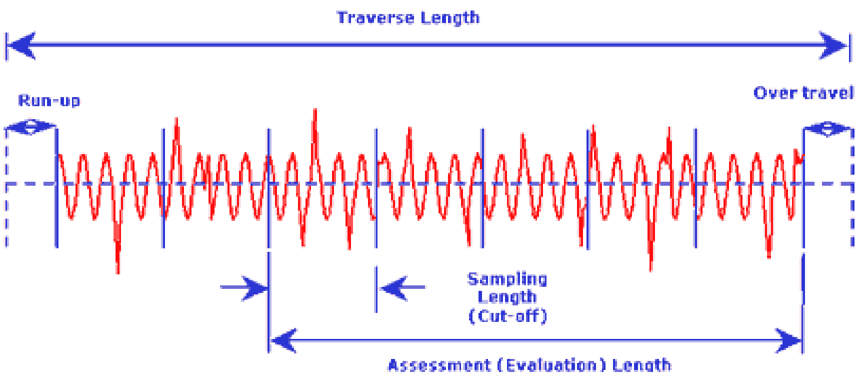


Figure 5.14: Example of traverse length, evaluation length and Sampling length.

Therefore, a class category "*Extraction*" can be defined with internal objects: num_cutoff, sampling_length, and evaluation_length.

(4) **Surface profile filtration**. The feature operation − filtration is used to separate the surface profile into roughness profile and waviness profile. The feature information for filtration includes filter type, and transmission band. In this project, various filters such as "Gaussian", "2RC" were used. The transmission band consists of all required wavelengths, which is defined at the short wave length by a short wavelength filter (lower limit) and the long wavelength by a

long wavelength filter (upper limit) (ISO 1302, 2002 [95]). The Figure 5.15 demonstrates a transmission band for roughness profiles formed with a short wavelength filter *λs* and a long wavelength filter λc as well as a transmission band for waviness profiles formed with a short wavelength filter λc and a long wave length filter *λf* (ISO 4287, 1997 [96]).

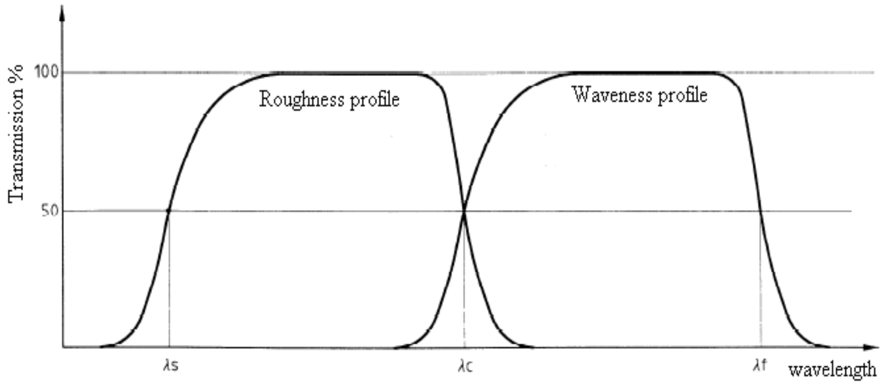


Figure 5.15: Examples of transmission band.

The band of sinusoidal profile wavelengths are transmitted at more than 50% when two phase correct filters of different cut-off wavelengths are applied to the profile. The transmission band shall be indicated by the inclusion of the cut-off values of the two filters (in millimetres), where the short-wave filter indicated at first and the long-wave follows the short one, and they are separated by a hyphen ("-"). For example, "0.0025-0.8" indicates a short-wave cut-off value of 0.0025 millimetres and a long-wave cut-off value of 0.8 millimetres, which will allow wave lengths between 0.0025mm and 0.8mm to be assessed with wavelengths below 0.0025mm and above 0.8mm being reduced in amplitude. Therefore, a class category "*Filtration*" can be defined with internal objects: filter_type, up_limit, and low_limit.

(5) **Comparison rule**. The comparison rule is used to compare the measured values with the tolerance values suggested by the Specification component, which determines whether the produced surface is within the tolerance. GPS includes two kinds of comparison rules: "max-rule" and "16%-rule". When the upper specification limit is used, the "16%-rule" indicates that the surface is considered acceptable if not more than 16% of all the measured values on an evaluation length exceed the tolerance value suggested by Specification. When the lower specification limit is used, the "16%-rule" indicates that the surface is considered acceptable if not more than 16% of all the measured values on an

evaluation length are less than the tolerance value suggested by Specification. The "max-rule" indicates that the surface is considered acceptable if none of the measured values for the suggested parameter over the entire surface exceed the tolerance value specified by specification. Therefore, a class category "*Comparison*" can be defined with internal objects: rule_type and rule_indication.

5.3.3 The Categorical Design Model Construction for Specification

In the analysis model construction stage, five analysis class categories have been defined. As stated before, in real applications, users often use default indications in the complete callout symbols. These default indications are inferred from the simple callout symbols by using GPS standards (e.g. "*Ra* 3.3"). Therefore, in the categorical design model construction stage, inference rules are added for these five class categories to form refined design classes. In the Specification component of VirtualGPS, the inference rules are a set of basic rules/constraints defined in the GPS standards, which uses some properties of a callout symbol to get other properties in the callout symbol. Furthermore, the complete callouts are actually "shallow" knowledge in the VirtualGPS system, which are used as illumination knowledge for inferring other "depth" knowledge in the Manufacture or Verification Component. The following paragraphs in this section focus on discussing these inference rules in or between these five class categories that form a complete callout symbol.

In the "*Measurand*" class category, there are two inference rules:

- **Inference rule for setting default tolerance types**. There are two types of tolerance limit for a surface, the upper tolerance limit and the lower tolerance limit. The indication can be of an upper type with indication "U" or of a lower type with indication "L" (ISO 1302, 2002 [95]). If not otherwise indicated, the default tolerance type is upper limit "U". Therefore, the inference rule for setting default tolerance types is represented as List 5.1:

```
RULE_NO 1
IF no value indicated for tolerance_type
THEN tolerance_type = "U"
```

List 5.1: Inference rule No.1.

This inference rule is applied on the internal object level, so it is represented as a method identity arrow mapping from the internal object "tolerance_type" to the same internal object "tolerance_type" of the "*Measurand*" class category.

- **Inference rule for setting default machining allowance**. The machining allowance is a planned deviation between an actual dimension and a nominal dimension, which is usually indicated only in those cases where more process stages are shown in the same drawing (ISO 1302, 2002 [95]). It allows an area of excess metal to be left to complete subsequent machining. The machining allowance is indicated in millimetres. Therefore, the inference rule for setting default machining_allowance is represented as List 5.2:

```
RULE_NO 2
 IF no value indicated for machining_allowance
THEN machining_allowance = NULL
```

List 5.2: Inference rule No.2.

This inference rule is applied on the internal object level, so it is represented as a method identity arrow mapping from the internal object "machining_allowance" to "machining_allowance" of the "*Measurand*" class category.

In the "*Partition*" class category, there are three inference rules for setting the default values for the direction symbol, manufacture type and manufacture method respectively. These three inference rules can be represented in the same way as the inference rule for setting the default tolerance type in "*Measurand*" class category. For example, the inference rule for setting default direction symbol is represented as List 5.3:

```
RULE_NO 3
IF no value indicated for direction_symbol
THEN direction_symbol = "Not Indicated"
```

List 5.3: Inference rule No.3.

Therefore, unless explicitly specified by users, the default values for direction symbol, manufacture type and manufacture method in the "*Partition*" are "Not Indicated". If manufacture method is not indicated in a complete callout generated by the Specification component, the Manufacture component can be used to determine suitable manufacturing processes matching the specification of the designed product (see Section 5.4).

In the "*Extraction*" class category, there are three inference rules:

- **Inference rule for setting default num_cutOff**. Two tables can be used to form the inference rule for setting default num_cutOff (see Table 5.4 and

Table 5.5). The Table 5.4 lists the indication of the number of sampling lengths for the three profile parameters (ISO 1302, 2002 [95]).

Profile	Num_cutOff indication
R-profile (roughness parameters)	If not otherwise indicated, the **default** number of cutOff wavelengths is 5 derived from ISO 4288 (ISO 4288, 1996 [99]). If the number of sampling lengths within the evaluation length differs from the default number of 5, it shall be indicated adjacent to the relevant parameter designation. For example *Rp3* or *Rv3* or *Rz3*..., *RSm3* ...all indicate that an evaluation length of 3 sampling lengths is desired.
W-profile (waviness parameters)	The number of sampling lengths shall always be indicated adjacent to the parameter designation of waviness. For example Wa3 or Wz3 ...all indicate that an evaluation length of three sampling lengths is desired.
P-profile (primary profile parameters)	The indication of the number of sampling lengths in the parameter designation of primary profile parameters is not relevant, as the evaluation length equals the sampling length and also equals the length of the feature being measured.

Table 5.4: Num_cutOff for profile parameters.

Table 5.5 lists the number of sampling lengths for parameters based on material ratio curve (ISO 13565-2, 1996 [97]; ISO 13565-3, 2000 [98]).

Profile	Num_cutoff indication
R-profile (roughness parameters)	1. If not otherwise indicated, the **default** number of cutoff wavelengths is 5 derived from ISO 13565-1 (ISO 13565-1, 1996 [100]). 2. If the number of sampling lengths within the evaluation length differs from the default number of 5, it shall be indicated adjacent to the relevant parameter designation. For example, *Rk*3 or *Rpk3* ...all indicate that an evaluation length of 3 sampling lengths is desired.
P-profile (primary profile parameters)	The indication of the number of sampling lengths in the parameter designation of primary profile parameters is not relevant, as the evaluation length equals the sampling length and also equals the length of the feature being measured.

Table 5.5: Num_cutOff for parameters based on material rate curve.

Based on Table 5.4 and 5.5, the inference rule for setting default num_cutOff is represented as List 5.4:

```
RULE_NO 4
IF  parameter_type ENDWITH a number
THEN num_cutOff = a number
ELSE IF no value indicated for num_cutOff
AND parameter_type STARTWITH "R"
THEN num_cutOff = 5
ELSE IF no value indicated for num_cutOff
AND parameter_type STARTWITH "P"
THEN num_cutoff=0
```

List 5.4: Inference rule No.4.

This inference rule is applied on the internal object level, so it is represented as a method identity arrow mapping from the internal object "num_cutOff" to the same internal object "num_cutOff" of "*Extraction*" class category.

- **Inference rule for setting default sampling length**. Table 5.6 lists values for the sampling length for profile parameters (ISO 1302, 2002 [95]).

Profile	Sampling length
R-profile	The sampling length may be indicated as the upper limit *λc* in the callout symbol *c*, see Figure 5.12. If there is no indication in the callout, tables 5.7 ~ 5.9 can be used to choose the roughness sampling length from the suggested parameter values, according to ISO 4288 (ISO 4288, 1996 [99]). For example take the surface parameter *Ra* with a limit value of 3.3 micrometers, according to table 5.7, the parameter value belongs to the range of 2 < *Ra* ≤ 10, and the related sampling length shall be 2.5 millimetres.
W-profile	There are no defaults for waviness sampling length given in ISO standards, so the sampling length is indicated as the upper limit in the callout symbol *c*, see Figure 5.12. For example, 0,8-25 / *Wz3* 10, the sampling length 25 millimetres is indicated as the upper limit in the callout symbol.
P-profile	In the default case, *P*-parameters do not have any sampling lengths. It may be indicated if required for the function of the workpiece where it is indicated as the upper limit in the callout symbol *c*, see Figure 5.12. For example -25 / *Pz* 225, the sampling length 25 millimetres is indicated.

Table 5.6: Default sampling lengths for profile parameters.

Based on Table 5.6, the inference rule for setting default sampling length with profile parameters is represented as List 5.5:

```
RULE_NO 5
IF parameter_category EQUALS "profile parameter"
AND  parameter_type STARTWITH "R"
AND no value indicated for sampling_length
THEN USE Inference Rule RULE_NO 6
ELSE IF parameter_category EQUALS "profile parameter"
AND  parameter_type STARTWITH "R"
AND up_limit has value up_limit
THEN sampling_length = up_limit
ELSE  sampling_length =NULL
ELSE IF parameter_category EQUALS "profile parameter"
AND parameter_type STARTWITH "W"
AND up_limit has value up_limit
THEN sampling_length = up_limit
ELSE  sampling_length =NULL
ELSE IF parameter_category EQUALS "profile parameter"
AND parameter_type STARTWITH "P"
AND up_limit has value up_limit
THEN sampling_length = up_limit
ELSE  sampling_length =NULL
```

List 5.5: Inference rule No.5.

A sub rule (rule No.6) is nested in the above rule, which is built upon the Table 5.7, 5.8 and 5.9.

Ra (µm)	Roughness sampling length Lr (mm)	Roughness evaluation length Ln (mm)
(0,000) < Ra ≤ 0,02	0,08	0,4
0,02 < Ra ≤ 0,1	0,25	1,25
0,1 < Ra ≤ 2	0,8	4
2 < Ra ≤ 10	2,5	12,5

10 < Ra ≤ 80	8	40

Table 5.7: Roughness sampling lengths for the measurement of *Ra*, *Rq*, *Rsk*, *Rku*, *RΔq* and curves and related parameters for non-periodic profiles.

Rz, Rz1max (µm)	**Roughness sampling length Lr (mm)**	**Roughness evaluation length Ln (mm)**
0,025 < Rz,Rz1max ≤ 0,1	0,08	0,4
0,1 < Rz,Rz1max ≤ 0,5	0,25	1,25
0,5 < Rz,Rz1max ≤ 10	0,8	4
10 < Rz,Rz1max ≤ 50	2,5	12,5
50 < Rz,Rz1max ≤ 200	8	40
1) Rz is used when measuring Rz, Rv, Rp, Rc and Rt 2) Rz1max is used when measuring Rz1max, Rv1max, Rp1max and Rc1max		

Table 5.8: Roughness sampling lengths for the measurement of *Rz*, *Rv*, *Rp*, *Rc* and *Rt* of non-periodic profiles.

RSm (µm)	**Roughness sampling length Lr (mm)**	**Roughness evaluation lengthLn (mm)**
0,013 < RSm ≤ 0,04	0,08	0,4
0,04 < RSm ≤ 0,13	0,25	1,25
0,13 < RSm ≤ 0,4	0,8	4
0,4 < RSm ≤ 1,3	2,5	12,5
1,3 < RSm ≤ 4	8	40

Table 5.9: Roughness sampling lengths for the measurement of *R*-parameters of periodic profiles, and *RSm* of periodic and non-periodic profiles.

The inference rule for Table 5.7 is given below and other two tables can be defined in same way as List 5.6 shown.

```
RULE_NO 6
IF  parameter_category EQUALS "profile parameter"
AND parameter_type Equals "Ra"
AND 0.000<parameter_value<=0.02
THEN sampling_length = 0.08
ELSE IF  parameter_category EQUALS "profile parameter"
AND parameter_type Equals "Ra"
AND 0.02<parameter_value<=0.1
THEN sampling_length = 0.25
ELSE IF parameter_category EQUALS "profile parameter"
 AND parameter_type Equals "Ra"
AND 0.1<parameter_value<=2
THEN sampling_length = 0.8
ELSE IF  parameter_type Equals "Ra"
AND parameter_category EQUALS "profile parameter"
AND  2<parameter_value<=10
THEN sampling_length = 2.5
ELSE IF  parameter_category EQUALS "profile parameter"
AND parameter_type Equals "Ra"
AND 10<parameter_value<=80
THEN sampling_length = 8
```

List 5.6: Inference rule No.6.

The motif parameters do not use the concept of sampling length. The operator used to calculate motif parameters has its own limit values, so sampling length concepts do not exit (ISO 12085, 1996 [25]). Table 5.10 lists the default value for the sampling length for parameters based on material ratio curve (ISO 13565-2, 1996 [97]; ISO 13565-3, 2000 [98]).

Profile	Sampling length
R-profile	If not otherwise indicated, the default sampling length for parameters based on material ratio curve is 0,8 millimetres derived from ISO 13565-1 (ISO 13565-1, 1996 [100]).
P-profile	In the default case, *P*-parameters do not have any sampling lengths. The sampling length equals the evaluation length and also equals the length of the feature being measured.

Table 5.10: Sampling lengths for parameters based on material ratio curve.

Based on Table 5.10, the inference rule for setting the default sampling length for parameters based on material ratio curve is represented as List 5.7:

```
RULE_NO 7
IF  parameter_category EQUALS "material ratio curve"
AND no value indicated for sampling_length
AND parameter_type STRATWITH "R"
THEN sampling_length = 0.8
ELSE IF  parameter_category EQUALS "material ratio curve"
AND no value indicated for sampling_length
AND parameter_type STRATWITH "P"
THEN sampling_length = evaluation_length
```

List 5.7: Inference rule No.7.

The inference rules (number 5, 6) are applied on category level between "*Measurand*" and "*Extraction*", so a pullback construct "determine_sampling_length" is built to contain these two inference rules (see Figure 3.26 of Chapter 3). As the inference rule − number 7 may also be used for parameters based on material ratio curve, so the pullback construct "*determine_sampling_length*" is typed with "*optional*".

- **Inference rule for setting default evaluation length**. The Table 5.11 lists default values of the evaluation length for profile parameters (ISO 1302, 2002 [95]).

Profile	Evaluation length
R-profile	If not otherwise indicated, the default length of the feature for roughness analysis consists of five sample lengths, so the evaluation length equals the num_cutoff x sampling_length. i.e. evaluation_length = num_cutoff x sampling length For example, take the surface parameter *Ra* with a limit value of 3.3 micrometers, i.e. *Ra* 3,3, according to table 5.7, the sampling length is 2.5 millimetres, and num_cutoff uses the default value 5, therefore, the evaluation length for this parameter is 5 x 2.5 = 12.5 millimetres.
W-profile	The default evaluation length of the waviness profile equals the num_cutoff x sampling length of the waviness profile. i.e. evaluation length = num_cutoff x sampling length For example, 0,8-25 / Wz3 10, the num_cutoff is indicated as 3 adjacent to the parameter designation Wz, and the sampling length 25 millimetres is indicated as the upper limit in the callout symbol, therefore, the evaluation length is 3 x 25 = 75 millimetres.
P-profile	For primary profiles, the evaluation length equals the sampling length and also equals the length of the feature being measured. i.e. evaluation length = sampling length For example, -25 / Pz 225, the evaluation length equals the sampling length of 25 millimetres as indicated in the callout.

Table 5.11: Evaluation lengths for profile parameters.

For motif parameters, the default evaluation length is 16 millimetres. Table 5.12 lists default values of the evaluation length for parameters based on material ratio curve (ISO 13565-2, 1996 [97]; ISO 13565-3, 2000 [98]).

Profile	Evaluation length
R-profile	The evaluation length of the roughness profile equals the num_cutoff x sampling length of the roughness profile. The default num_cutoff of the roughness profile equals five and the default sampling length of the roughness profile is 0,8 millimetres. i.e. evaluation length = num_cutoff x sampling length
P-profile	For primary profiles, the evaluation length equals the sampling length which is also equal to the length of the feature being measured. i.e. evaluation length = sampling length

Table 5.12: Evaluation lengths for parameters based on material ratio curve.

Based on the Table 5.11, 5.12 and the default evaluation length for motif parameters, the inference rule for setting evaluation length is represented as List 5.8:

```
RULE_NO 8
IF parameter_category EQUALS “profile parameter”
AND no value indicated for evaluation_length
AND parameter_type STRATWITH “R”
THEN evaluation_length = 5× 0.8
ELSE IF parameter_category EQUALS “profile parameter”
AND no value indicated for evaluation_length
AND parameter_type STRATWITH “W”
THEN sampling_length = num_cutOff ×sampling_length
ELSE IF parameter_category EQUALS “profile parameter”
AND no value indicated for evaluation_length
AND parameter_type STRATWITH “P”
```

```
THEN evaluation_length = sampling_length
ELSE IF  parameter_category EQUALS "motif parameter"
AND no value indicated for evaluation_length
THEN evaluation_length = 16
ELSE IF  parameter_category EQUALS "material ratio curve"
AND no value indicated for evaluation_length
AND parameter_type STRATWITH "R"
THEN evaluation_length = 5× 0.8
ELSE IF  parameter_category EQUALS "material ratio curve"
AND no value indicated for evaluation_length
AND parameter_type STRATWITH "P"
THEN evaluation_length = sampling_length
```

List 5.8: Inference rule No.8.

This inference rule is applied on internal object level, so it is represented as two method arrows mapping from internal object "sampling_length" and "num_cutOff" to "evaluation_length" respectively in "*Extraction*" class category. However, the default evaluation_length is not always determined by "sampling_length" and "num_cutOff" (e.g. motif parameter), so these two method arrows are optional.

In "*Filtration*" class category, there are four inference rules:

- **Inference rule for setting default upper and lower limit of the transmission band**. For profile parameters, the cut-off value of the upper limit equals to the sampling length. Therefore, the inference rule for setting the default upper limit of a transmission band with profile parameters is represented as List 5.9:

```
RULE_NO 9
IF  parameter_category EQUALS "profile parameter"
AND no value indicated for up_limit
THEN up_limit = sampling_length
```

List 5.9: Inference rule No.9.

For motif parameters, two bounds *A* and *B* are used in the motif algorithms according to ISO 12085 for defining the maximum widths of the roughness and waviness motifs respectively. The width for the roughness motif *ARj* should be greater than the value of *λs* and less than or equal to *A*, see Figure 5.16.

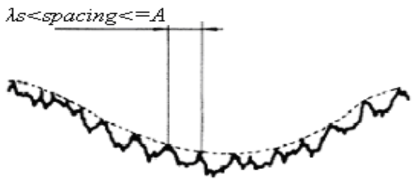


Figure 5.16: Roughness motifs.

The width for the waviness motif *AWj* should be greater than the value of *A* and less than or equal to the value of *B*, see Figure 5.17.

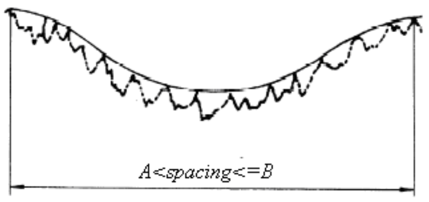


Figure 5.17: Waviness motifs.

The *A* and *B* can be obtained from Table 5.13 according to the evaluation length.

Evaluation length (mm)	**A (mm)**	**B (mm)**	**λs (µm)**
0,64	0,02	0,1	2,5
3,2	0,1	0,5	2,5
16	0,5	2,5	8
80	2,5	12,5	25

Table 5.13: Transmission band for motif parameters.

Based on the Table 5.13, the inference rule for setting upper limit with motif parameters is represented as List 5.10:

```
RULE_NO 10
IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND evaluation_length=0.64
THEN up_limit = 0.02
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND evaluation_length=3.2
THEN up_limit = 0.1
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND evaluation_length=16
THEN up_limit = 0.5
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND evaluation_length=80
THEN up_limit = 2.5
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND evaluation_length=0.64
THEN up_limit = 0.1
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND evaluation_length=3.2
THEN up_limit = 0.5
ELSE IF parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND evaluation_length=16
THEN up_limit = 2.5
ELSE IF parameter_category EQUALS "motif parameter"
```

```
AND parameter_type EQUALS "AWj"
AND evaluation_length=80
THEN up_limit = 12.5
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type NOT EQUALS "ARj" OR "AWj"
THEN up_limit = NULL
```

List 5.10: Inference rule No.10.

For parameters based on material ratio curve, the upper limit λc is defined as equal to the sampling length according to ISO 1302 and ISO 13565-1 (ISO 1302, 2002 [95]; ISO 13565-1, 1996 [100]). As the default sampling length for parameters based on material ratio curve is 0.8 millimetres (refer to the inference rule for setting default sampling length with parameters based on material ratio curve), the inference rule for setting upper limit with parameters based on material ratio curve is represented as List 5.11:

```
RULE_NO 11
IF  parameter_category EQUALS "material ratio curve"
AND no value indicated for up_limit
THEN up_limit = 0.8
```

List 5.11: Inference rule No.11.

Therefore, the up limit for the transmission band can be defined in two optional ways: 1. for motif parameters, the up limit is defined by evaluation length and parameter type; 2. for profile parameters and parameters based on material ratio curve, the up limit is defined by sampling length. In the first case, a 3-ary pullback construct "determine_up/low_limit" among "*Measurand*", "*Extraction*" and "*Filtration*" is built to contain the corresponding inference rule (see Figure 3.27 of Chapter 3). In the second case, a pullback construct "*equals*" between "*Extraction*" and "*Filtration*" is built to contain the corresponding inference rule (see Figure 3.25 of Chapter 3). These two pullback constructs are all typed in optional.

Table 5.14 lists the values of the lower limit for profile parameters (ISO 1302, 2002 [95]).

Profile	Lower limit
R-profile	Lower limit λs may be indicated as the lower limit in the callout symbol *c*, see Figure 5.12. If there is no indication in the callout, lower limit *λs* can be obtained from ISO 3274 according to the value of upper limit *λc*, seeTable 5.15 (ISO 3274, 1996 [101]).
W-profile	The lower limit of the W-profile transmission band is *λc* (short-wave filter), and will be indicated as the lower limit in the callout symbol *c*, see Figure 5.12.
P-profile	The lower limit of the P-profile of the transmission band is λs (short-wave filter), and will be indicated as the lower limit in the callout symbol *c*, see Figure 5.12.

Table 5.14: Lower limit for profile parameters.

λc (mm)	λs (μm)	λc/λs	r_{tip} max (μm)	Maximum sampling spacing
0,08	2,5	30	2	0,5
0,25	2,5	100	2	0,5
0,8	2,5	300	2	0,5
2,5	8	300	5	1,5
8	25	300	10	6

Table 5.15: Relationship between the roughness cut-off wavelength *λc*, tip radius and roughness cut-off ratio *λc/ λs*.

Based on the Table 5.14 and 5.15, the inference rule for setting lower limit with profile parameters is represented as List 5.12:

```
RULE_NO 12
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND no value indicated for lower_limit
THEN USE inference Rule RULE_NO 13
ELSE IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "W"
AND no value indicated for lower_limit
THEN lower_limit = NULL
ELSE IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "P"
AND no value indicated for lower_limit
THEN lower_limit = NULL
```

List 5.12: Inference rule No.12.

The inference rule for Table 5.15 is given below as List 5.13:

```
RULE_NO 13
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND up_limit = 0.08
THEN lower_limit = 0.0025
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND up_limit = 0.25
THEN lower_limit = 0.0025
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND up_limit = 0.8
THEN lower_limit = 0.0025
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND up_limit = 2.5
THEN lower_limit = 0.008
IF  parameter_category EQUALS "profile parameter"
AND parameter_type STRATWITH "R"
AND up_limit = 8
THEN lower_limit = 0.025
```

List 5.13: Inference rule No.13.

Table 5.16 lists the value of the lower limit for motif parameters.

Profile	Lower limit
R profile	As mentioned in Figure 5.16, the width for the roughness motif *ARj* should be greater than the value *λs* according to ISO 12085 (ISO 12085, 1996 [25]). The lower limit *λs* can be obtained from Table 5.13 according to the evaluation length.
W profile	As mentioned in Figure 5.17, the width for the waviness motif *AWj* should be greater than the value *A* according to ISO 12085 (ISO 12085, 1996 [25]). The lower limit *A* can be obtained from Table 5.13 according to the evaluation length.

Table 5.16: Lower limit for motif parameters.

Based on the Table 5.16, the inference rule for setting lower limit for motif parameters is represented as List 5.14:

```
RULE_NO 14
IF  parameter_category EQUALS "motif parameter"
AND parameter_type STRATWITH "R"
AND no value indicated for lower_limit
THEN USE Inference Rule RULE_NO 15
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type STRATWITH "W"
AND no value indicated for lower_limit
THEN USE Inference Rule RULE_NO 16
```

List 5.14: Inference rule No.14.

The inference rule of number 15 based on Table 5.13 is represented as List 5.15:

```
RULE_NO 15
IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND no value indicated for lower_limit
AND evaluation_length=0.64
THEN lower_limit = 0.0025
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND no value indicated for lower_limit
AND evaluation_length=3.2
THEN lower_limit = 0.0025
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND no value indicated for lower_limit
AND evaluation_length=16
THEN lower_limit = 0.008
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "ARj"
AND no value indicated for lower_limit
AND evaluation_length=80
THEN lower_limit = 0.025
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND no value indicated for lower_limit
AND evaluation_length=0.64
THEN lower_limit = 0.02
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND no value indicated for lower_limit
AND evaluation_length=3.2
THEN lower_limit = 0.1
ELSE IF  parameter_category EQUALS "motif parameter"
```

```
AND parameter_type EQUALS "AWj"
AND no value indicated for lower_limit
AND evaluation_length=16
THEN lower_limit = 0.5
ELSE IF  parameter_category EQUALS "motif parameter"
AND parameter_type EQUALS "AWj"
AND no value indicated for lower_limit
AND evaluation_length=80
THEN lower_limit = 2.5
ELSE IF parameter_category EQUALS "motif parameter"
AND no value indicated for lower_limit
AND parameter_type NOT EQUALS "ARj" OR "AWj"
THEN lower_limit = NULL
```

List 5.15: Inference rule No.15.

Table 5.17 lists the values of the lower limit for parameters based on material ratio curve.

Profile	Lower limit
R profile	If not otherwise indicated, the default lower limit λs for roughness profiles is 0,0025 millimetres according to ISO 1302 and ISO 13565-1 (ISO 1302, 2002 [95]; ISO 13565-1, 1996 [100])
P profile	The lower limit for primary profiles of the transmission band is λs (short-wave filter), which has no default value to be defined according to ISO 1302 (ISO 1302, 2002 [95]).

Table 5.17: Lower limit for parameters based on material ratio curve.

Based on the Table 5.17, the inference rule for setting lower limit with parameters based on ratio curve is represented as List 5.16:

```
RULE_NO 16
IF  parameter_category EQUALS "material ratio curve"
AND parameter_type STRATWITH "R"
AND no value indicated for lower_limit
THEN lower_limit=0.0025
IF  parameter_category EQUALS "material ratio curve"
AND parameter_type STRATWITH "W" or "P"
AND no value indicated for lower_limit
THEN lower_limit=NULL
OR lower_limit= NOT INDICATED
```

List 5.16: Inference rule No.16.

Therefore, inference rules (No.12, 13, 14, and 15) for setting the default low limit of transmission band are applied on the internal object level, so it is represented as a method arrow mapping from internal object "up_limit" to "low_limit" in "*Filtration*" class category. This method arrow is optional. The inference rule No.16 is represented as a indentity arrow mapping from internal object "lower_limit" to the same internal object "lower_limit" of "*Filtration*" class category.

- To finally form a complete callout, an inference rule should be built as List 5.17:

RULE_NO 17
IF QUERYING Callout
AND Getting all default values for Partition
AND Getting all default values for Extraction
AND Getting all default values for Filtraction
AND Getting all default values for Measurand
AND Getting all default values for Comparison
THEN Generating Callout

List 5.17: Inference rule No.17.

This inference rule is actually the procedure knowledge for generating complete callouts. The complete callout can be regarded as shallow knowledge that is used for reasoning other deeper knowledge. Besides adding these inference rules in these class categories, a set for functional dependencies arrows should also be added in the design model construction stage. For example, in "*Measurand*" class category, the internal object "parameter_name" is functional dependent on "parameter_type" to provide the complete parameter names. So a functional dependency arrow is mapping from "parameter_type" to "parameter_name". After adding all these inference rules and functional dependencies in or between class categories, the categorical diagrams for graphically representing the structured knowledge in Specification component can be devised as Figure 3.25, 3.26, 3.27 and 3.28 of Chapter 3.

5.3.4 The Categorical Sequence Diagram Construction for Specification

The sequence diagram for Specification component is used to clarify the process to generate a complete callout from a simple callout on drawing, see Figure 5.18.

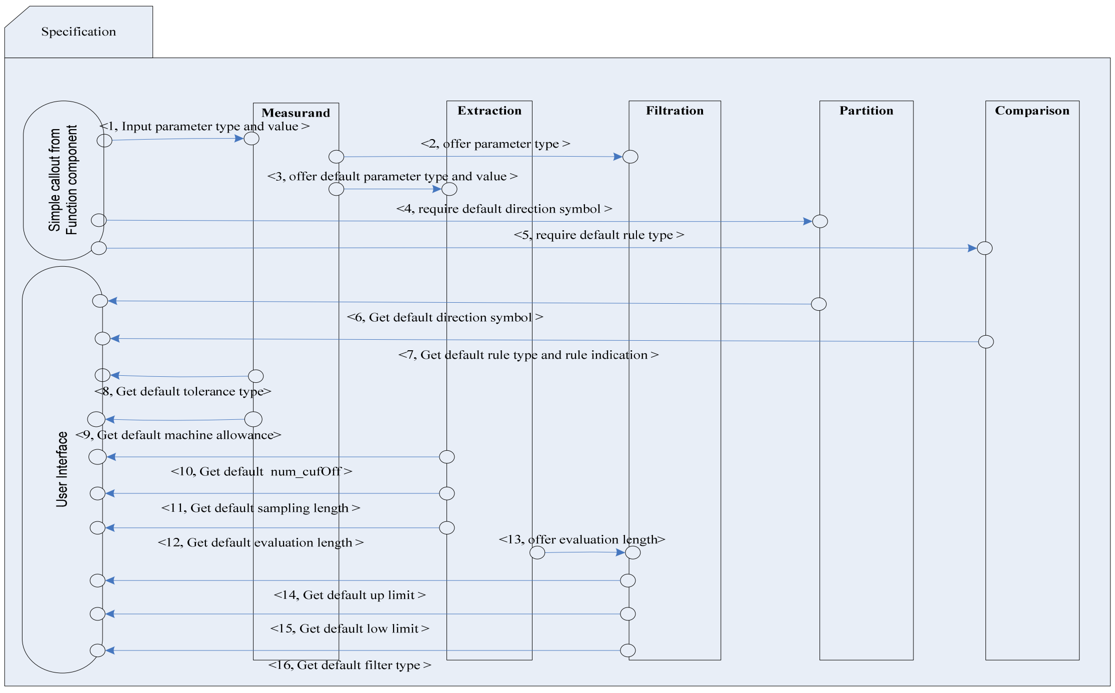

Specification
Measurand
Extraction
Filtration
Partition
Comparison
Simple callout from Function component
User Interface
<1, Input parameter type and value >
<2, offer parameter type >
<3, offer default parameter type and value >
<4, require default direction symbol >
<5, require default rule type >
<6, Get default direction symbol >
<7, Get default rule type and rule indication >
<8, Get default tolerance type>
<9, Get default machine allowance>
<10, Get default num_cufOff >
<11, Get default sampling length >
<12, Get default evaluation length >
<13, offer evaluation length>
<14, Get default up limit >
<15, Get default low limit >
<16, Get default filter type >

Figure 5.18: The sequence diagram for Specification component.

5.3.5 The Categorical deployment model Construction for Specification

Based on the Figure 5.18, a deployment topological graph for Specification can be devised as Figure 5.19.

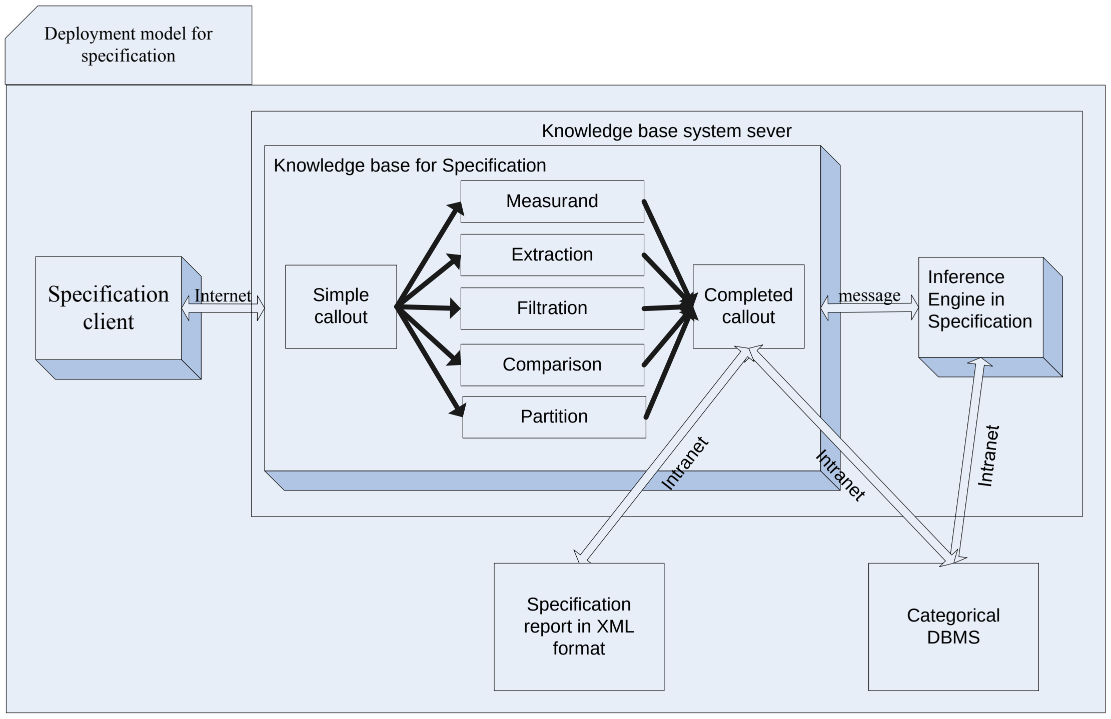

Deployment model for specification
Knowledge base system sever
Knowledge base for Specification
Measurand
Extraction
Filtration
Comparison
Partition
Simple callout
Completed callout
Specification client
Internet
message
Inference Engine in Specification
Intranet
Intranet
Intranet
Specification report in XML format
Categorical DBMS

Figure 5.19: The deployment topological graph for Specification component.

Figure 5.19 shows how to allocate difference design classes in Specification component on computing resource nodes, and how these design classes interacted with other modules or components such as user interfaces and the categorical DBMS. An inference engine should be built to control the inference rules defined for the knowledge base in Specification, which also needs to communicate with the database to retrieve existing complete callouts for users. The complete callout can be outputted in XML format as a specification report for different user communications.

5.4 The Manufacture Component Design

This section aims to provide a detailed discussion of the design of the Manufacture component, which focuses on discussing the knowledge acquisition and knowledge organization. The Manufacture component is used to help users to select suitable manufacturing processes matching the callout specification provided by Specification component.

5.4.1 The Categorical Business Map Construction for Manufacture

Manufacture is a transformation process from raw material into finished products, which includes the design of the product, the selection of raw materials and the sequence of processes (manufacturing processes) through which the product will be made (Kalpakjian and Schmid, 2005 [92]). The manufacturing process is one of the core parts for manufacturing industry. Generally, a manufacturing process can be classified into six groups in general: casting processes, moulding processes, forming processes, machining processes, joining process and rapid manufacturing (Schey, 2000 [102]). Since there are various kinds of processes, the selection of a suitable process becomes a difficult job for new manufacturers. Therefore, the main software functions for the Manufacture component can be summarized as:

- Generate suitable comprehensive manufacturing processes by organizing a set of sub-processes for users.
- Provide a broad overview for each suggested process through giving detailed information including material suitability, design considerations, quality issues, general economics, process fundamental and variations. This overview should be organized in a standard format.

Based on these two software functions, a business map for Manufacture component can be developed as Figure 5.20 illustrated.

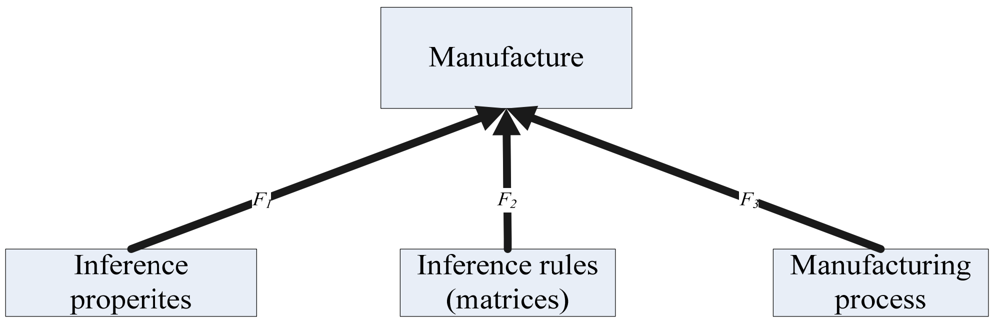


Figure 5.20: The business map for Manufacture component.

The Figure 5.20 shows that the Manufacture component needs to contain three use cases: a set of inputted inference properties, a set of inference rules based on several matrices and a class category for holding all stored manufacturing processes.

5.4.2 The Categorical Analysis Model Construction for Manufacture

In this section, the analysis model is used to separate these three use cases defined in Figure 5.20 into main class categories (analysis classes) as the following three points demonstrate:

(1) **Inference properties**. The inference properties can be separated into two groups: specification related properties and PRIMA (Manufacturing Process

Information Map) related properties. The specification related properties include the texture lay, surface parameter type, tolerance value and cut-off wavelength, which can be obtained from the specification report (complete callout) provided by Specification part. Two PRIMA selection matrices are used in Manufacture based on two properties: material type and production quantity. Therefore, two class categories should be devised to represent these two groups – "*SpecificationProperties*" and "*PRIMAProperties*".

(2) **Inference rules**. According to the discussion in point one, two groups of inference properties indicate two groups of inference rules. The PRIMA selection matrices are used for PRIMA related properties. This is a simple inference method based on material and production quantity, which is designed to enable users to focus their attentions on the most relevant PRIMAs. For instance, a complete PRIMA matrix used in the Manufacture component can be referred in Appendix *E* (Swift and Booker, 2003 [103]). On the other hand, the surface texture is also an important issue that needs to be considered when selecting the manufacturing processes.

- **Texture lay**. Texture lay is the directionality of the surface, which is an important factor affecting the interaction between the surface and the environment. Table 5.18 lists some examples of typical manufacturing processes suitable to different texture lays (Griffiths, B., 2001 [104]).

Lay symbol	**Interpretation**	**Typical Manufacturing processes**
$=$ $\perp$	Parallel to plane of projection of view in which symbol is used Perpendicular to plane of projection of view in which symbol is used	milling, drilling, turning, shaping
X	Crossed in two oblique directions relative to plane of projection of view in which symbol is used	cross-honing
M	Multi-directional	lapping, abrading
C	Approx. circular relative to centre of surface to which symbol applies	facing, parting-off
R	Approx. radial relative to centre of surface to which symbol applies	face-grinding
P	Lay is particulate, non-directional, or protuberant	EDM, ECM, peening

Table 5.18: Texture lay with typical manufacturing processes.

- **Surface roughness values**. A typical manufacturing process has the ability to produce a limited range of surface roughness values *Ra*, between 1.6

µm – 6.3 µm, see Table 5.19 (BS 1134-2, 1990 [105]).

Key: ■ average application ▨ less frequent application

Process	Roughness values (µm *Ra*)												
	50	25	12.5	6.3	3.2	1.6	0.8	0.4	0.2	0.1	0.05	0.025	0.0125
Flame cutting	▨	■	▨										
Snagging	▨	■	■	▨									
Sawing	▨	■	■	■	▨								
Planing, shaping	▨	■	■	■	■	■	▨						
Drilling			▨	■	■	▨							
Chemical milling			▨	■	■	▨							
			▨	▨■	■	▨							
Electro-discharge		▨	▨	■	■	■	▨	▨					
Broaching				▨	■	■	▨						
Reaming				▨	■	■	▨						
Boring, turning		▨	▨	■	■	■	■	▨	▨	▨			
Barrel finishing					▨	▨	■	■	▨	▨			
Electrolytic grinding							▨■	■	▨				
							▨	■	▨				
Roller burnishing				▨	▨	■	■	■	■	▨	▨		
Grinding					▨	■	■	■	▨	▨			
Polishing							▨	■	■	▨	▨	▨	
Lapping							▨	■	■	■	▨	▨	
Superfinishing							▨	▨	■	■	▨	▨	
Sandcasting	▨	■	▨										
Hot rolling	▨	■	▨										
Forging		▨	■	■	▨								
Permanent				▨	■	▨							
Investment casting				▨	■	▨	▨						
			▨	▨	■	■	▨						
Extruding				▨	■	■	▨	▨					

Cold rolling, drawing					▨	■	▨						
Note: The ranges shown above are typical of the processes listed. Higher or lower values may be obtained under special conditions.													

Table 5.19: Surface roughness values produced by common production processes and materials.

- **Cut-off wavelength**. The cut-off wavelength is used to distinguish roughness values and waviness values. Table 5.20 shows the suitable cut-off wavelength for different manufacturing processes (Leach, 2001 [106]).

	Cut-off wavelength (mm)				
Process	**0.25**	**0.8**	**2.5**	**8.0**	**25.0**
Milling		√	√	√	
Turing		√	√		
Grinding	√	√	√		
Shaping		√	√	√	
Boring		√	√	√	
Planning			√	√	√
Reaming		√	√		
Broaching		√	√		
Diamond boring	√	√			
Diamond turning	√	√			
Honing	√	√			
Lapping	√	√			
Super finishing	√	√			
Buffing	√	√			
Polishing	√	√			
Electro discharge	√	√			
Burnishing		√	√		
Drawing		√	√		
Extruding		√	√		
Moulding		√	√		
Electro polishing		√	√		

Table 5.20: Choice of cut-off wavelength for a number of common machining operations.

The Tables 5.18, 5.19 and 5.20 become class categories "*CriteriaOne*", "*CriteriaTwo*" and "*CriteriaThree*". These three class categories can manage all inference information defined in Table 5.18, 5.19 and 5.20 and hold the inference methods.

(3) **Manufacturing process**. In order to assist users in making final decisions after getting a set of suggested manufacturing processes through matrices and tables defined in the previous point, a standard format is required to represent each manufacturing process. The PRIMA format defined by Swift and Booker

has been chosen to achieve this requirement. The PRIMA format is a deliberate standard format, which gives detailed information on the characteristics and capabilities of each process with specific headings including: material suitability, design considerations, quality issues, general economics and process fundamentals and variations. Therefore, in order to support the manufacturing process report rendered in the PRIMA format, a class category "*ManufacturingProcess*" was devised to record all information relating to a manufacturing process for PRIMA with following internal objects (Swift and Booker, 2003 [103]):

- **Process Description**: an explanation of the fundamentals of the process together with a diagrammatic representation of its operation. (e.g. the drilling is a process that removal of material by chip processes using rotating tools of various types with two or more cutting edges to produce cylindrical holes in a workpiece)
- **Materials**: describes the materials currently suitable for the given process (e.g. the materials suitable for the drilling process are all metals and some plastics and ceramics).
- **Process Variations**: a description of any variations of the basic process and any special points related to those variations (e.g. wide ranges of cutting tool materials are available for the drilling process).
- **Economic Considerations**: a list of several important points − production rate, minimum production quantity, tooling costs, labour costs, lead times and any other points which may be of specific relevance to the process (e.g. the tooling costs and finishing costs for the drilling process are low).
- **Typical Applications**: a list of components and parts that have been successfully manufactured using the process (e.g. one of the typical applications for the drilling process is any component requiring cylindrical holes).
- **Design Aspects**: any points, opportunities or limitations that are relevant to the design of the part as well as standard information on minimum section, size range and general configuration (e.g. flat-bottomed holes should be avoided for the drilling process).
- **Quality Issues**: standard information includes a process capability chart,

surface roughness, as well as any information on possible faults, etc (e.g. surface roughness values ranging 0.4 – 12.5 μm *Ra* are obtainable for the drilling process).

5.4.3 The Categorical Design Model Construction for Manufacture

This section refines the six analysis classes determined in Section 5.4.2 by adding relationships and constraints between them. Based on the Figure 3.23 that shows the coequalizer construct for reasoning the suitable manufacture procedures, the diagram Figure 5.21 can be defined to model design classes involved in the Manufacture component in detail.

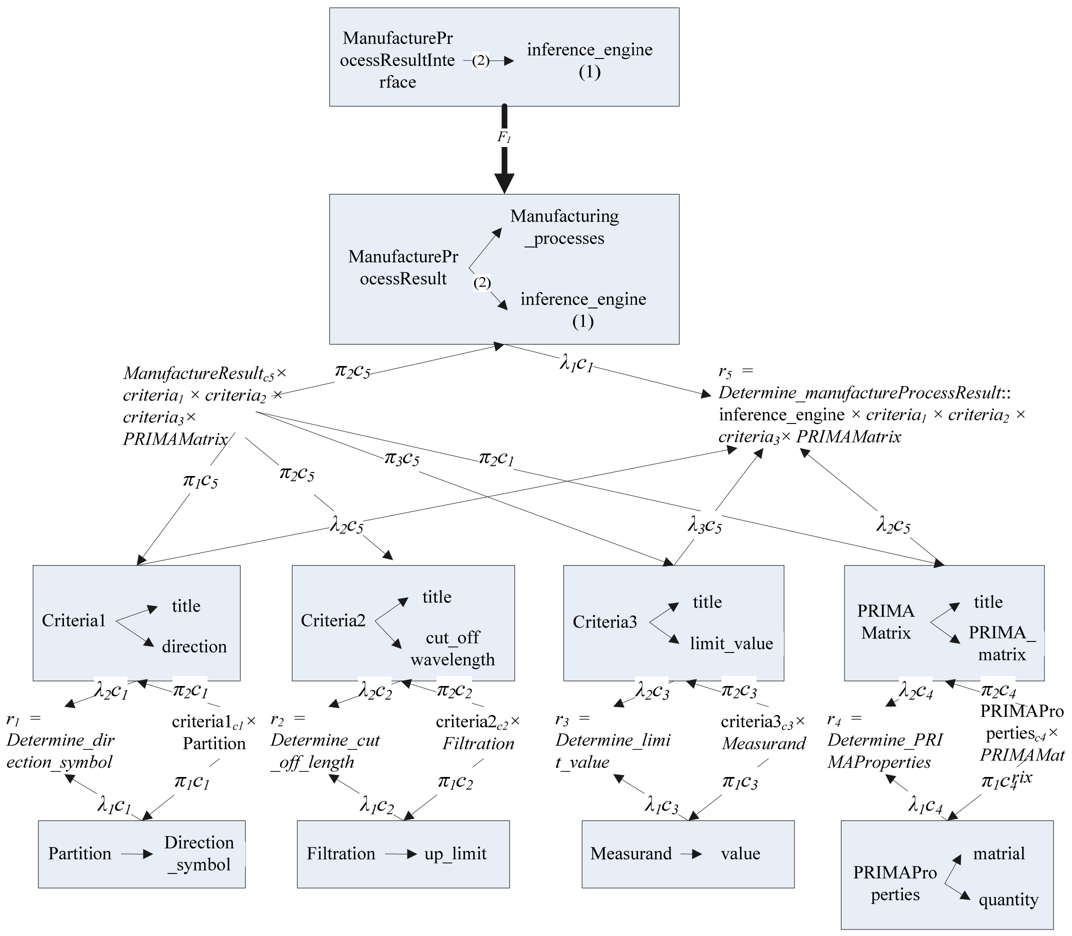


Figure 5.21: The categorical object model for Manufacture component.

Figure 5.21 shows a categorical object modelling diagram used to represent class categories and their relationships relating to the Manufacture component. This categorical object model contains a 5-ary relationship that specifies the criteria1, criteria2, criteria3 and PRIMA matrix are working together with “inference_engine” method to infer the suitable manufacturing processes. Besides managing inference rules defined in three criteria and the PRIMA matrix, the “inference_engine” also contains an algorithm to calculate the weight of each suggested manufacturing process, see equation 5.1.

$W_1 = \mu_{Criteria1}(c) + \mu_{Criteria2}(c) + \mu_{Criteria3}(c)$

$\{c \in \textit{manufacturing processes set in VirtualGPS}\}$ (5.1)

List 5.18: Equation 5.1.

In equation 5.1, if a manufacturing process *c* can be got through criteria$_1$, the $\mu_{Criteria1}(c) = 0.3$, otherwise $\mu_{Criteria1}(c) = 0.0$. The same result is applied to $\mu_{Criteria2}$(c) and $\mu_{Criteria3}(c)$. The W_1 indicates the weight value for a manufacturing process after inferred by *criteria$_1$*, *criteria$_2$* and *criteria$_3$*, which is an intersection value for $\mu_{Criteria1}(c)$, $\mu_{Criteria2}(c)$ and $\mu_{Criteria3}(c)$. For PRIMA matrix, the weight calculation is achieved by equation 5.2.

$W_2 = \sigma_{PRIMAMatrix}(c)$

$\{c \in \textit{manufacturing processes set in VirtualGPS}\}$ (5.2)

List 5.19: Equation 5.2.

In equation 5.2, if a manufacturing process c can be got through PRIMA matrix, the $\sigma_{PRIMAMatrix}$(c) = 0.5, otherwise $\sigma_{PRIMAMatrix}$(c) = 0.0. The Manufacture component considers five top weights for W_1 and five top weights for W_2 together to get final five top weight manufacturing processes that will be rendered to users initially.

5.4.4 The Categorical Sequence Diagram Construction for Manufacture

The sequence diagram in Manufacture is used to clarify the process to generate PRIMA reports for top five weight manufacturing processes is shown in Figure 5.22.

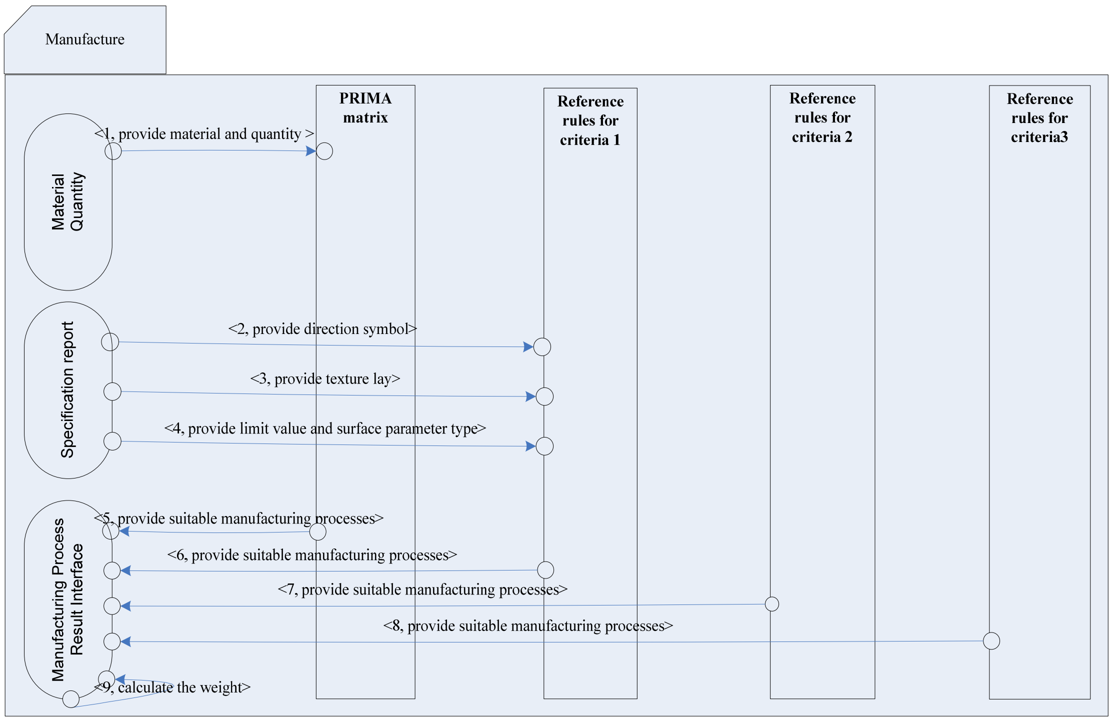

Manufacture
PRIMA matrix
Reference rules for criteria 1
Reference rules for criteria 2
Reference rules for criteria3
Material Quantity
Specification report
Manufacturing Process Result Interface
<1, provide material and quantity >
<2, provide direction symbol>
<3, provide texture lay>
<4, provide limit value and surface parameter type>
<5, provide suitable manufacturing processes>
<6, provide suitable manufacturing processes>
<7, provide suitable manufacturing processes>
<8, provide suitable manufacturing processes>
<9, calculate the weight>

Figure 5.22: The sequence diagram for Manufacture component.

Figure 5.22 further detail the reasoning process of Figure 3.23 through a set of sequential arrows.

5.4.5 The Categorical Deployment Model Construction for Manufacture

Based on the Figure 5.22, a deployment topological graph for Manufacture can be devised as shown in Figure 5.23.

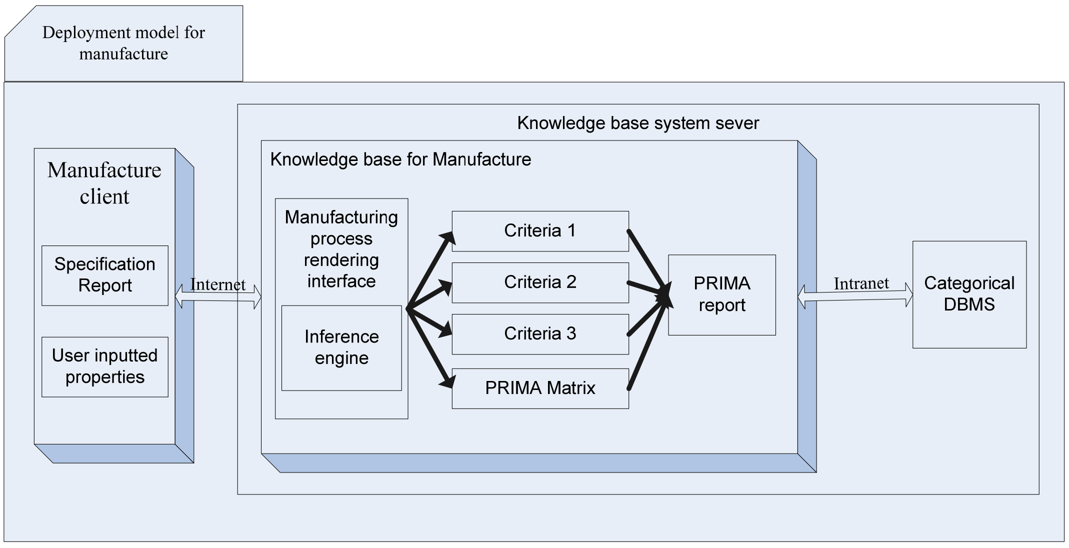


Figure 5.23: The deployment topological graph for Manufacture component.

Figure 5.23 shows how to allocate difference design classes in the Manufacture component on computing resource nodes and how these design classes interacted with other modules or components (e.g. Specification component) and the categorical DBMS. The inference engine in "*ManufacturingProcessResultInterface*" class category is responsible for inferring suitable manufacturing processes, calculating weight values and formatting them into PRIMA format for users.

5.5 The Verification Component Design

This section aims to provide a detailed discussion for the design of the Verification component, which focuses on discussing the knowledge acquisition and knowledge organization. The Verification component is used to determine the verification procedures: to select an appropriate measuring instrument for determining how to obtain the features from real surfaces; to suggest how to calculate the measured parameter value; and to compare the measured value with the tolerance value.

5.5.1 The Categorical Business Map Construction for Verification

A complete measurement procedure should contain: instrument chosen, partition, extraction, filtration, parameters to be calculated and comparison rule. For verification of a manufactured product, the following steps should be taken:

(1) Getting a set of surface texture specifications for the manufactured product, which includes surface texture parameters and their tolerance values.

(2) Choosing a suitable instrument to match the measuring requirements defined in specifications.

(3) Calculating values for measured parameters generated by suggested filters.

(4) Comparing measured values of suggested surface texture parameters with the tolerance values corresponding to these suggested surface texture parameters.

Therefore, verification contains two parts: measurement procedure and comparison process. The Verification component in VirtualGPS system aims to cover these two parts:

- Measurement procedure in the Verification component contains: defining traverse length, defining filtering technique, selecting measurement instruments.
- Comparison process in the Verification component contains: defining comparison rules.

The detailed software functions of Verification component can be summarised as follows:

(1) Refer the measurement procedure definitions and contents. Users can obtain

the detailed explanation of each operation within the measurement procedure, such as the definition of traverse length, the sampling spacing and etc.

(2) Generate a suitable measurement procedure, including traverse length, traverse direction, sampling length, cut-off wavelength of filters, filter type, instrument and comparison rule.

(3) Infer suitable candidate measurement instruments for users.

(4) Check detailed characteristics of candidate instruments. Users can carry out further comparison of suggested instruments and make a final decision.

(5) Provide the comparison result after inputting both the measurand and the measured value. The system can calculate the result by using certain comparison rules and determine whether the surface is within the tolerance.

(6) Further refer to the Function, the Specification and the Manufacture sub-knowledge bases. The Verification component is connected with the others and users can easily traverse through them.

Based on analysis above, the categorical business map for the Verification component can be built as Figure 5.24 shown.

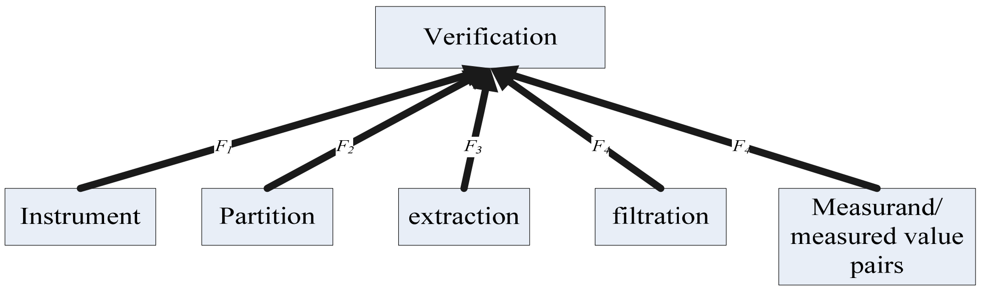


Figure 5.24: The business map for Verification component.

The Figure 5.24 shows that the Verification component contains five use cases: "*Instrument*", "*Partition*", "*Extraction*", "*Filtration*", and "*Measurand/measured values pairs*".

5.5.2 The Categorical Analysis Model Construction for Verification

The five use cases defined in Figure 5.24 are refined into class categories (analysis classes) in this section through following five points:

(1) **Partition**. Since surface texture is influenced by the detailed form of the profile curve, the feature information needed for carrying out the partition must include the traverse length of the surface profile being evaluated and the traverse direction of the measurement instrument. The traverse length is the length of surface traversed by the measurement instrument and the traverse

direction is the direction traced by the measurement instrument during a measurement (Leach, 2001 [106]). The traverse direction should be perpendicular to the direction of the surface texture lay unless otherwise indicated. Therefore, the "*Partition*" class category defined in the Specification component (see Section 5.3.3) is also used in Verification, but adds the items of two other internal objects: traverse_length and traverse_direction.

(2) **Extraction**. In the Verification component, lower limit, sampling spacing and sampling length are used to identify a finite number of measuring points from the surface. To obtain the lower limit, if no default value is indicated in Specification component, the Verification component will ask users to input values for lower limit. The sampling spacing is the width length between two adjacent measuring points on the surface, which can be obtained from ISO 3274 according to the value of upper limit *λc* or lower limit *λs* (see Table 5.21) (ISO 3274,1996 [101]).

λc **mm**	**λs** **µm**	**Maximum sampling spacing** **µm**
0,08	2,5	0,5
0,25	2,5	0,5
0,8	2,5	0,5
2,5	8	1,5
8	25	5

Table 5.21: Relationship between the roughness cut-off wavelength *λc* and maximum sampling spacing.

Based on Table 5.21, the inference rule for setting default sampling spacing is represented as List 5. 20:

```
RULE_NO 18
IF  λc = 0.08
THEN sampling_spacing  = 0.0005
ELSE IF  λc = 0.25
THEN sampling_spacing  = 0.0005
ELSE IF  λc = 0.8
THEN sampling_spacing  = 0.0005
ELSE IF  λc = 2.5
THEN sampling_spacing  = 0.0015
ELSE IF  λc = 8
THEN sampling_spacing  = 0.005
```

List 5.20: Inference rule No.18.

To obtain a sampling length: if no default value is indicated in the Specification component, the Verification component will ask users to provide values for sampling length (e.g. motif parameters do not use the concept of sampling length). After capturing the knowledge above, the "*Extraction*" class category defined in Section 5.3.3 is also used in the Verification component. However, the "*Extraction*" class category is refined in Verification through adding an internal object "sampling_spacing" with an inference rule No.18. The inference rule No.18 is depicted in Figure 5.27 by adding a method arrow mapping from "low_limit" to "sampling_spacing" and this method arrow is optional. Other structures and inner- or inter-relationships/inference rules of "*Extraction*" class category defined in the Specification component are preserved in the Verification component.

(3) **Filtration**. In the verification stage, the filter type and cut-off wavelength are used for guiding users to separate the surface profile into a roughness profile and a waviness profile. The cut-off wavelength is used as a means of separating or filtering the wavelengths of a surface. The value of cut-off wavelength is equal to the upper limit defined in "*Filtration*". Therefore, the "*Filtration*" class category defined in the Specification is also be used in the Verification with all inference rules and relationships preserved. For example, getting the default value for the cut-off wavelength is same as getting the default for the upper limit defined in Specification through using inference rules defined for upper limit (see Section 5.3.3).

(4) **Measurand/measured value pairs**. The Measurand and value pairs are modelled using categorical object model as shown in Figure 3.22. Figure 3.22 also shows how knowledge is structured and interacted in a comparison process. The detailed modelling of a measurement procedure can be referred in Section 3.6 of Chapter 3.

(5) **Instrument**. Measurement of surface topography plays an important role in manufacturing, which is used for both control of manufacturing processes and for determining whether the final product is acceptable or not. More importantly, in the modern manufacturing industry, the measurement of surface topography can also help manufacturers improve their product designs. Different measurement procedures can be performed using different instruments which have different capabilities and limitations (ISO 13565-1, 1996 [100]). There are three groups of instruments:

- The stylus instruments. Stylus instruments are contact instruments, which use styluses as the central components of the probes.

- The optical instruments. Optical instruments use the optical probes, and involve projecting light on to a surface. They are non-contact instruments.
- Other instruments that include the new generation of scanning microscopes such as the Scanning Electron Microscope (SEM), the Scanning Tunnelling Microscope (STM) and the Atomic Force Microscope (AFM) (Whitehouse,1997 [107]). They use scanning probes that utilize electrons rather than light.

The instruments selection is required to match instrument attributes with measuring requirements. In real applications, the measurement range and resolution of different instruments are the most important factors that need to be considered. The A-W diagram is used to help the selection of instruments by defining an amplitude-wavelength plot, see Figure 5.25.

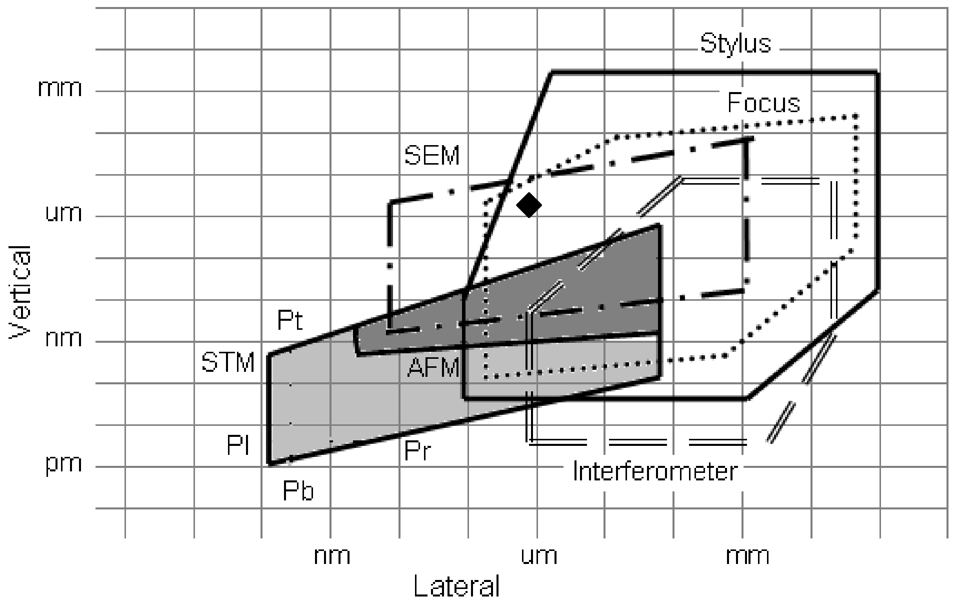


Figure 5.25: Selection of a measurement instrument.

In Figure 5.25, the vertical axis represents the resolution while the lateral axis represents the range of the instruments. For example, the ◆ symbol in Figure 5.25 illustrates the *Ra* 3.3, the horizontal coordinate of which can be located by the sampling spacing, and the vertical coordinate can be located by the parameter value. In this case, after inferred by VirtualGPS system, the sampling space for *Ra* 3.3 is 1.5μm and the parameter value is 3.3 μm, so an A-W plot can be defined as (1.5 μm, 3.3 μm) in Figure 5.25. According to the A-W diagram, three instruments have the capability of carrying out the measurement for *Ra* 3.3: Stylus, Focus and SEM. After getting candidate instruments from A-W diagram, the Verification component can provide a reference table (Table 5.22) for users to check the detailed characteristics of these candidate instruments. After which users can choose the most suitable instrument for the measurement (Whitehouse, 1997 [107]).

Method	**Measurement tool**	**Spatial resolution**	**Spatial range**	**Z resolution**	**Range z**	**Frequency**	**Comments**
Stylus	Stylus tip	0.1μm	100mm	0.3nm	1000μm	20Hz	Contacts workpiece
Focus	Optical probe	0.5μm	50mm	0.5nm	100μm		Non-contacting
Interferometer	Optical probe	1μm	10mm	0.01nm	10μm	minutes	Non-contacting
SEM	Detection	0.01μm	1mm	2nm	10μm	minutes	Vacuum needed
STM	Conductive probe	0.0001μm	0.1mm	0.001nm	0.1μm	minutes	Only for the conducting surfaces
AFM	Atom force tip	0.005μm	0.08mm	1nm	0.1μm	minutes	Both for conducting and non conducting surfaces

Table 5.22: The characteristics for typical instruments.

The system can also allow users to insert new instruments into the knowledge base of Verification. Users are required to add the new instruments with essential attributes: *z* resolution, *z* range, spatial resolution and spatial range. The system can automatically generate a new polygon on the A-W diagram for a new instrument according to its attributes, and insert a new row in the characteristics table for it as well. Therefore, based on the analysis above, a class category "*Instrument*" should be defined as Figure 5.26.

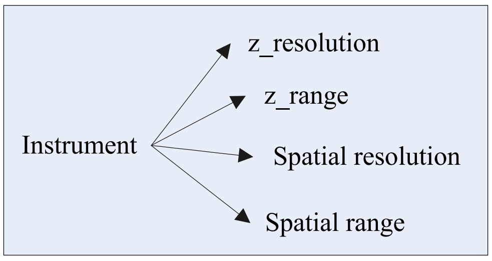


Figure 5.26: Categorical representation for "*Instrument*" class category.

5.5.3 The Categorical Design Model Construction for Verification

This section refines five analysis classes determined in Section 5.5.2 through adding relationships and constraints between them (see Figure 5.27). Figure 5.27 shows a categorical object model for the Verification component which contains five class categories: "*Partition*", "*Extraction*", "*Filtration*", "*Instrument*" and "*ComparingSquare*".

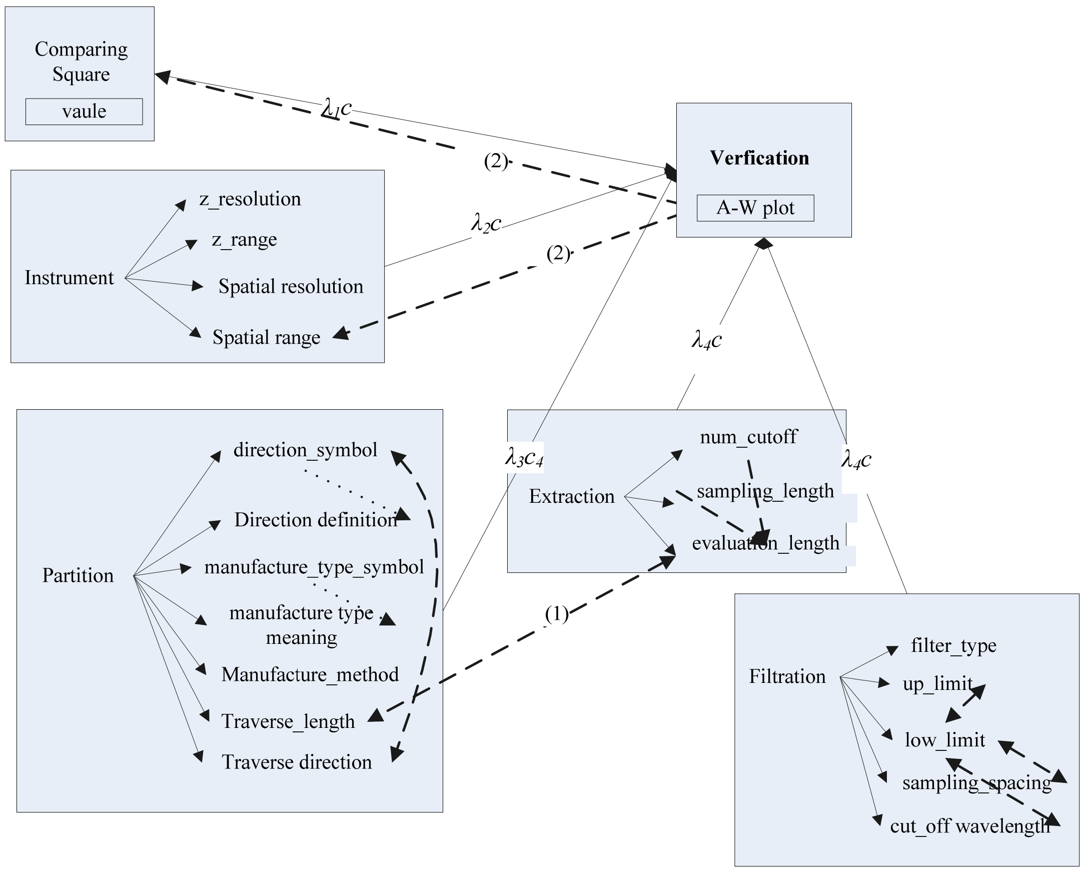


Figure 5.27: The categorical object model for Verification component.

In Figure 5.27, the "$\lambda_X C$" is a 5-ary pullback relationship which is used to represent the organizing and rendering of knowledge such as instrument suggestions, comparing results, interfaces for inputting new instruments, and measured values based on five class categories. The arrow in dashed line with number *(2)* indicates an inference rule for determining A-W plot using sampling spacing and the value of a suggested surface texture parameter. This inference rule is represented as a pullback relationship in Figure 5.27. The arrow in dashed line with number *(1)* is a constraint to specify that the "traverse_length" in "*Partition*" should be greater than the "evaluation_length" in "*Extraction*". The three pullback relationships in Figure 5.27 can be constructed in same way as pullback relationships defined for the Specification in Section 3.7.2.

Furthermore, according to the Figure 3.21 of Chapter 3, mirror relationships exist between Specification and Verification. Therefore, pullback relationships defined among "*Partition*", "*Extraction*", "*Filtration*", "*Comparison*" and "*Measurand*" in Specification should also be preserved in Verification.

5.5.4 The Categorical Sequence Diagram Construction for Verification

The detailed explanation for constructing a sequence diagram for Verification is specified in Section 3.5.1 of Chapter 3. The "*Comparison*" class category contains comparison rules and their instructions that can be used to guide specific comparison processes.

5.5.5 The Categorical Deployment Model Construction for Verification

The detailed explanation for constructing a deployment topological graph for Verification is specified in Section 3.5.2 of Chapter 3. The "*MeasurementProcedureManager*" in Figure 3.19 is responsible for generating suitable measurement procedure reports for users. The measurement procedure reports are generated based on the mirror relationships of class categories in specification reports.

5.6 Implementation of the VirtualGPS System

This section starts with a brief explanation on tools and platform for implementation of the VirtualGPS system. It then moves on to demonstrate how to use XML DOM + XSLT to dynamically generate reports such as function report or manufacture report for users. This section concludes with a test case analysis to assess the design functions of the system.

5.6.1 Tools and Platform for Developing the VirtualGPS

As the VirtualGPS is a distributed Java project, the following tools are used in this project:

- Java 2 SDK (Java for Software Developer Kit) in version 1.4.2.10. The Java 2 SDK contains: Java Compiler, Java Virtual Machine, Java Class Libraries, Java AppletViewer, Java Debugger, and other tools, which supports compiling and running Java program on Microsoft Windows (Sun, 2007 [108]).
- The Eclipse Platform in version 3.2. The Eclipse Platform is designed for building integrated development environments (IDEs). It can be used to create diverse end-to-end computing solutions for multiple execution environments

(Erickson and McIntyre, 2001 [109]). In this project, the Eclipse platform is used to help programmer in developing the VirtualGPS faster and easier. It can also benefit to organize different modules and software components of VirtualGPS in a unified framework.

- Standard Widget Toolkit (SWT) plug-in for Eclipse − swt_win32.jar (Eclipse, 2007 [110]). This plug-in is used to visually develop the graphic user interfaces.
- JfreeChar.jar plug-in for Eclipse (JFreeChart, 2007 [111]). This plug-in is used to dynamically draw various charts and diagrams for the VirtualGPS system.

The detailed introduction on how to set up a Java project using Eclipse can be found in tutorials published on the Eclipse official web site (Eclipse, 2005 [112]).

5.6.2 XML/XSLT Reports

In this project, XML is widely used in organizing various reports for users. XML was firstly defined in 1998 (XML 1.0) recommended by the World Wide Web Consortium (W3C) (Harold, 2002 [80]). At present, XML is the most widely used data interchange technique for holding structured data and controlling data communication. As XML is not designed to specify the rendering of data information as HTML did, XSLT is used to transform an XML file into another text-based form such as HTML pages that can be browsed on client screens. In order to generate a report, such as a function report in the VirtualGPS system, the following steps should be adopted:

(1) Querying or inferring knowledge from VirtualGPS.

(2) Formatting knowledge into XML files. The List 5.21 illustrates a XML file that is automatically generated by the system after querying function patterns from the categorical DBMS.

```
<?xml-stylesheet type="text/xsl" href="functionPattern.xsl"?>
 <root>
  <surfaceTexture.FunctionPattern InternalId="16889">
  <id>3</id>
 <patternid>pattern3</patternid>
 <componentName>Cylinder Liner</componentName>
  <context>The designers need to select the suitable specification for a surface in order
              to ensure the surface functions correctly.
 </context>
  <problem>Determination of the surface parameters to satisfy Pattern 2 - Functional
               performance of the surface.
 </problem>
 <solution>There are two basic approaches: 1.          Establish Pattern
                  4 - Functional correlation with texture parameters; 2.        First
                  establish a stable surface generation process that produces acceptable
                  surfaces and then Pattern 5 - Monitor for surface changes.
 </solution>
```

```
    <forces>The functional correlation approach is superior in quality of results but is
              more expensive in time and cost to establish correlation and more
              sophisticated measuring equipment is required than establishing a stable
              surface generation process and monitoring for surface changes.
    </forces>
    <example>The surface requirements for a cylinder liner on an engine block are that it
                needs to have a good bearing surface but also retain a reservoir of oil for
                lubrication.  1. The texture parameters Rk and friends have been shown to
                have a functional correlation with the desired surface tasks.  2.        One
                approach for manufacture is with a plateau-honed surface. Rq &amp; Rsk
                can be used to monitor for surface changes.
    </example>
    <nextPattern>After the surface parameters selection, try Pattern 4 - Functional
                  correlation and Pattern 5 - Monitor for surface changes.
    </nextPattern>
  </surfaceTexture.FunctionPattern>
</root>
```

List 5.21: A function report in XML format.

(3) Using XLST to transform XML into HTML. The List 5.22 illustrates a XLST file that is used to render function report based on patterns specified in Section 3.7.1.

```
<?xml version = "1.0"?>
<!-- functionPattern1.xsl -->
<!-- XSLT stylesheet for transforming content generated by -->
<!-- GetProductServlet into XHTML-->
<xsl:stylesheet version = "1.0" xmlns:xsl =
    "http://www.w3.org/1999/XSL/Transform">
 <xsl:output method = "xml" omit-xml-declaration = "no"
    indent = "yes" doctype-system = "DTD/xhtml1-strict.dtd"
    doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>

 <xsl:template match = "root">
  <html xmlns = "http://www.w3.org/1999/xhtml" xml:lang = "en" lang = "en">
   <head>
    <link rel = "StyleSheet" href = "CSS.css"/>
    <title>Pattern 1(Surface Requirements)<xsl:value-of select
              ="surfaceTexture.FunctionPattern/componentName"/>
</title>
  </head>

  <body>
   <div class="div3">
    <table border="2" cellpadding="0" cellspacing="0" class="table2"
     bordercolorlight="#ffffff" bordercolordark="#ffffff">
     <tr>
      <td  class="table3" bordercolorlight="#000000"
         bordercolordark="#ffffff" colspan="2">
       <p class="p1">=== Surface Requirements ===</p>
      </td>
     </tr>
     <tr>
      <td class="td12"><br/>Name: <xsl:value-of select =
                             "surfaceTexture.FunctionPattern/componentNa
                             me"/><br/><br/></td> </tr>
              …………………
     <td class="td7"><br/><xsl:value-of select =
            "surfaceTexture.FunctionPattern/nextPattern"/><br/><br/></td></tr>
    </table>
   </div>
   <br/>
  </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

List 5.22: An example of XLST codes for the function report.

The Figure 5.30 illustrates the final function report rendering for users. (See Section 5.6.3)

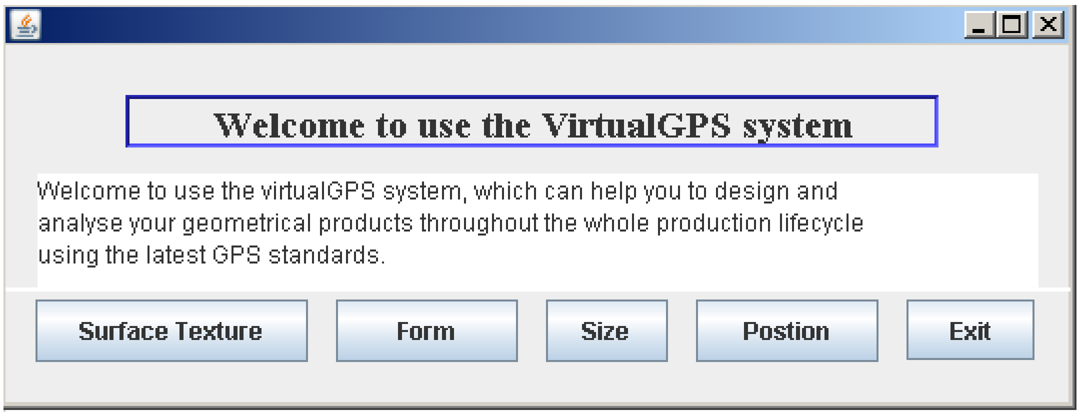
5.6.3 A Test Case Analysis for Cylinder Liner Design

As mentioned in previous sections, the VirtualGPS system can be used by designers to design products, and by metrologists to verify the design specifications. This case study analyses and demonstrates the design process for a cylinder liner. A cylinder liner is one of the central working parts of a reciprocating engine, and it is the space in which a piston travels. The movement of a piston inside the cylinder can drive a vehicle moving. Normally, a piston moves inside each cylinder with several metal piston rings fitted around its outside surface in machined grooves —typically two for compressional sealing and one for oil sealing. They are commonly made of spring steel and have close contact with the hard walls of the cylinder bore, which rides on a thin layer of lubricating oil to prevent the engine from seizing up. The contact between the cylinder liner and its counterpart piston rings requires the cylinder to have a good bearing surface but also retain a reservoir of oil for lubrication. Furthermore, the space surrounded by the cylinder bore and piston rings need a tight seal to contain the compression of fuel and air mixtures.

Among all the design features, the most important functional demands on the cylinder and piston rings are oil consumption, blow-by, and wear; especially at the top-dead centre (TDC). The surface texture parameters defined in the latest GPS standards have direct influences on the functional performance of the cylinder and piston. After performing a factorial designed experiment (FDE) where surface roughness was correlated to important functional performance indicators − oil consumption, wear, and blow-by, in a 10−litre truck engine, it was proved that 'oil consumption' is strongly correlated to the *Rz* parameter measured on the cylinder liner. The biggest influence on 'blow-by' is the *Ra* parameter measured on the piston rings with a negative variation. The 'wear' is also strongly correlated to the *Ra* value measured on the piston rings, followed by the *Rz* measured on the cylinder; both have the same variation with the 'wear'.

To demonstrate the functionality and usage of the VirtualGPS system, the design of a cylinder liner is performed on the platform in the following steps:

Step 1: launching the system. Figure 5.28 shows a snapshot of the main user entry of the VirtualGPS.


Welcome to use the VirtualGPS system

Welcome to use the virtualGPS system, which can help you to design and analyse your geometrical products throughout the whole production lifecycle using the latest GPS standards.

Figure 5.28: The main user entry interface.

Step 2: choose the "Surface Texture" button to enter the specific function page, in this case, the Surface Texture module interface. By activating the "Classic Components" menu on the menu bar, users can select the "Cylinder Liner" from the list and then move on to the "Design" sub-menu item. From here, users can enter the cylinder liner design process, which is consisted of four stages. Figure 5.29 shows the main user interface for the Surface Texture module.

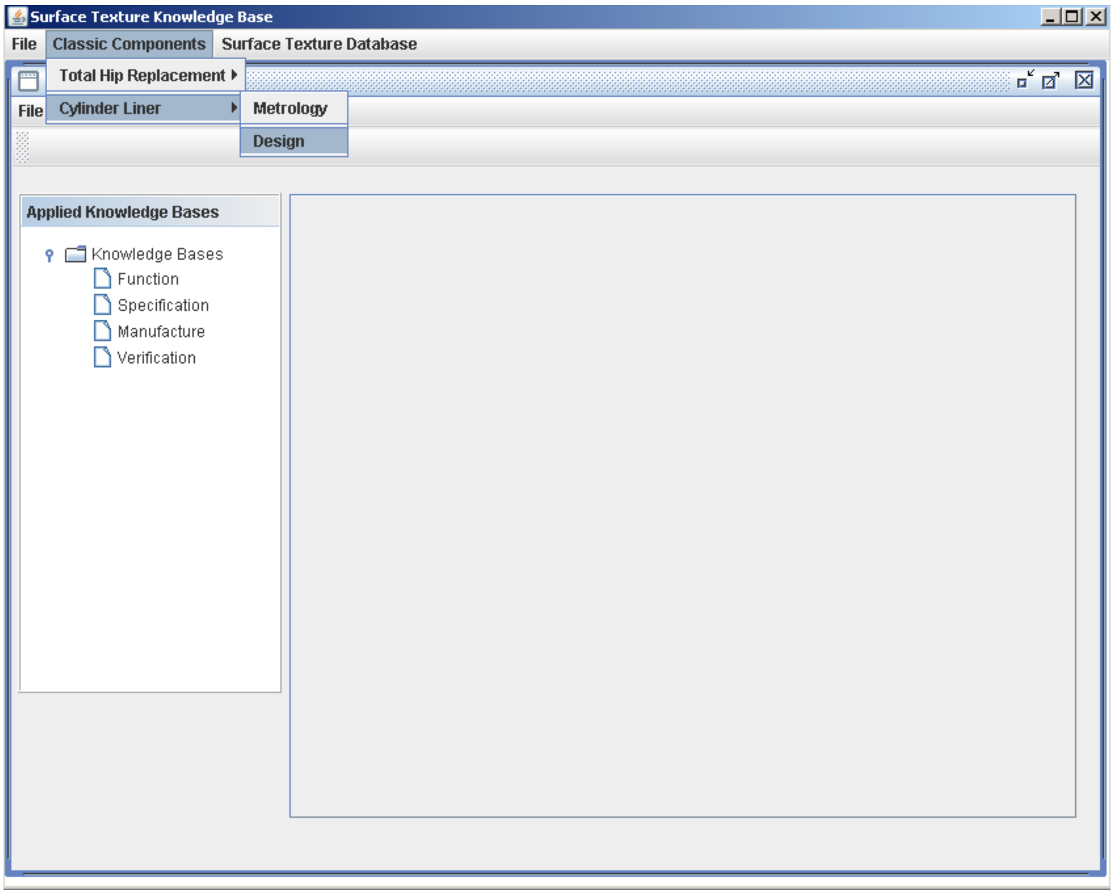


Figure 5.29: The Surface Texture working page.

Step 3: By double clicking the "*Function*" tree node on the Surface Texture page, a function analysis report is generated based on the calculation performed by the inference engine. It provides options for engineering designers with suitable surface parameters that match the required functional performances in the predefined patterns as shown below:

Pattern 1 — Surface requirements

For a cylinder liner on an engine block, the counterpart is the piston ring; the surface requirement is to maintain a good bearing surface while retaining a reservoir of oil for lubrication.

Pattern 2 — Functional performance

The most important functional demands in this case are correct oil consumption, blow-by, and wear especially at the top-dead centre (TDC).

Pattern 3 — Surface parameters selection

The texture parameters *Rk* and *Rz* have been shown to have a functional correlation with the desired surface tasks given in patterns 1&2. One option for the manufacturing process is to adopt a plateau-honed surface.

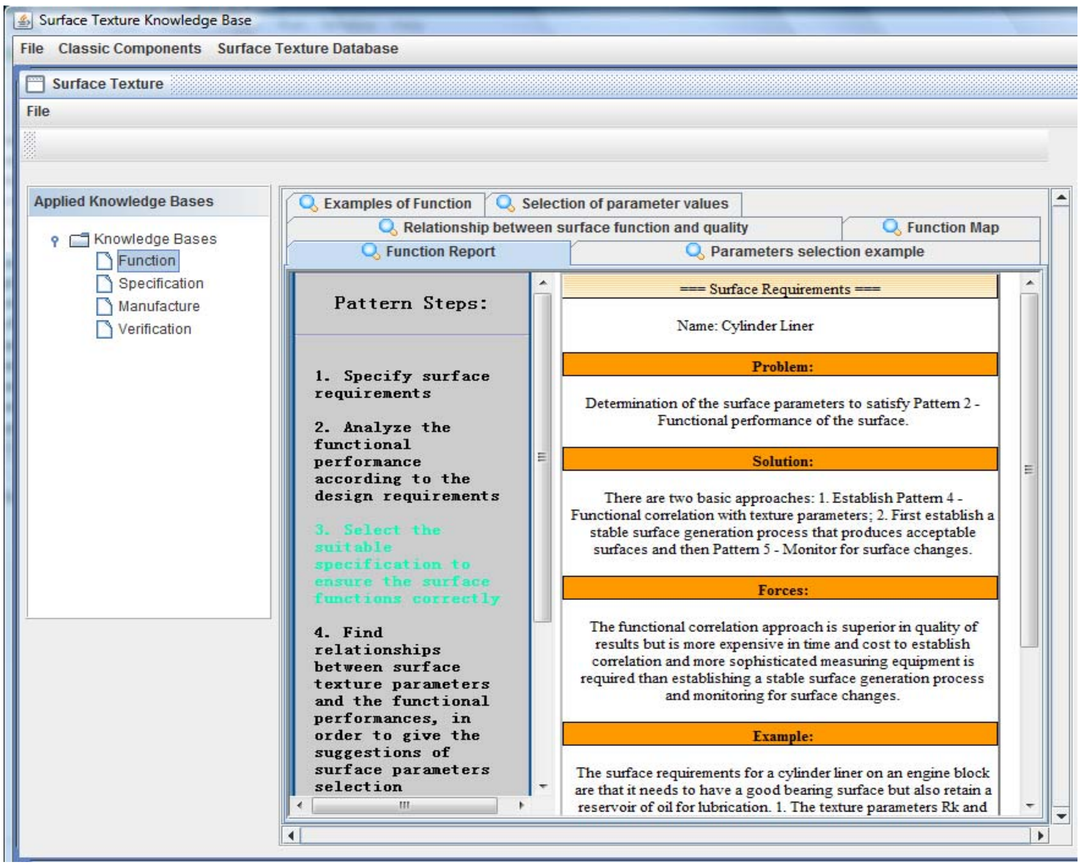
Rq & *Rsk* can be used to monitor for surface changes.

Pattern 4 — Functional correlation

The surface texture parameters *Rk* and *Rz* have been shown to have a functional correlation with the desired surface tasks.

Pattern 6 — Suggestion of limit values

According to the factorial designed experiment (FDE), when *Rz* of the cylinder increased, oil consumption, blow-up and wear all increased since they have the same variation. Therefore, the limit value of *Rz* is suggested at 4 μm in this case.



Surface Texture Knowledge Base
File Classic Components Surface Texture Database
Surface Texture
File
Applied Knowledge Bases
Knowledge Bases
Function
Specification
Manufacture
Verification
Examples of Function
Selection of parameter values
Relationship between surface function and quality
Function Map
Function Report
Parameters selection example

Pattern Steps:

1. Specify surface requirements
2. Analyze the functional performance according to the design requirements
3. Select the suitable specification to ensure the surface functions correctly
4. Find relationships between surface texture parameters and the functional performances, in order to give the suggestions of surface parameters selection

=== Surface Requirements ===

Name: Cylinder Liner

Problem:

Determination of the surface parameters to satisfy Pattern 2 - Functional performance of the surface.

Solution:

There are two basic approaches: 1. Establish Pattern 4 - Functional correlation with texture parameters; 2. First establish a stable surface generation process that produces acceptable surfaces and then Pattern 5 - Monitor for surface changes.

Forces:

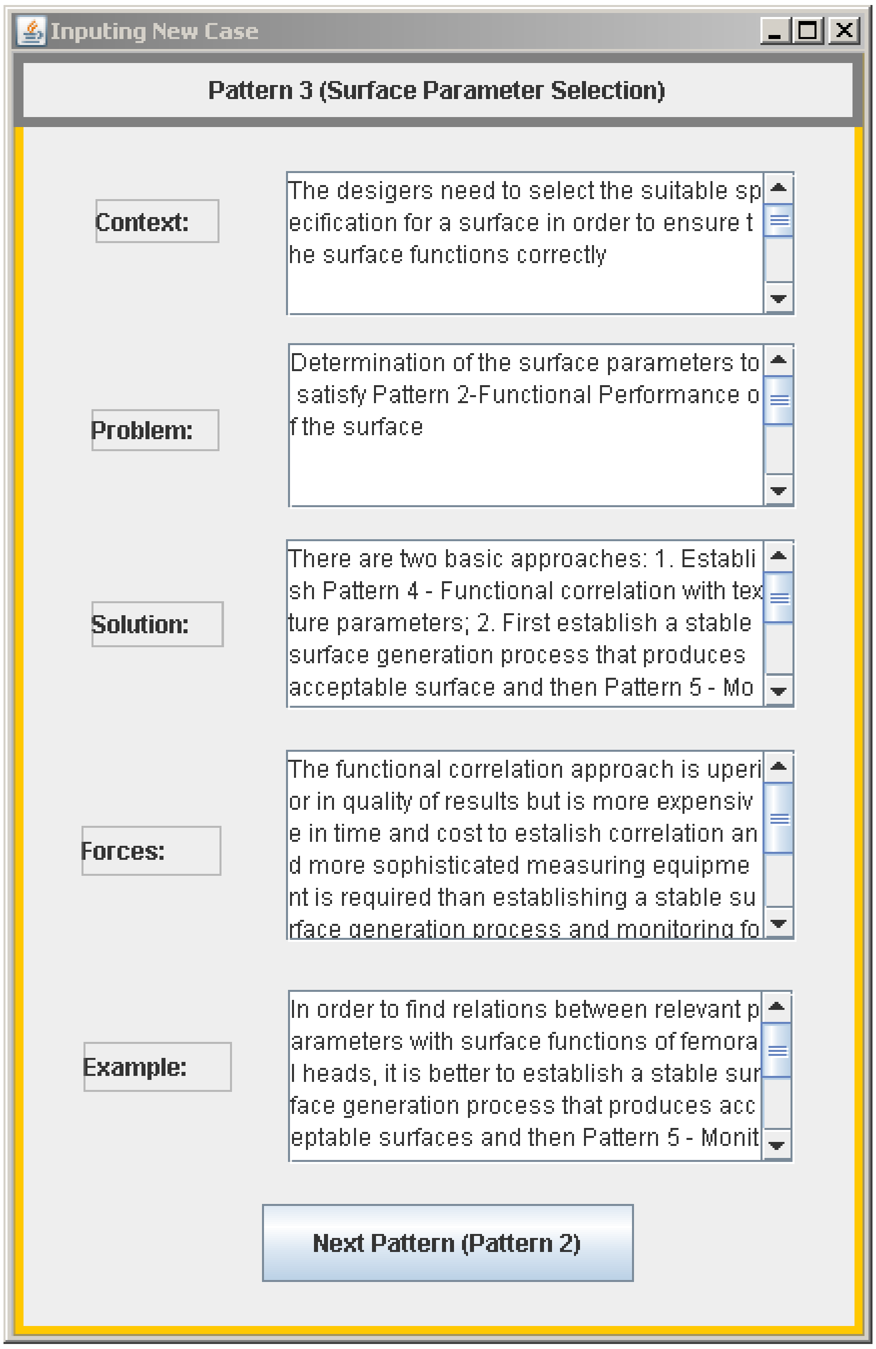
The functional correlation approach is superior in quality of results but is more expensive in time and cost to establish correlation and more sophisticated measuring equipment is required than establishing a stable surface generation process and monitoring for surface changes.

Example:

The surface requirements for a cylinder liner on an engine block are that it needs to have a good bearing surface but also retain a reservoir of oil for lubrication. 1. The texture parameters Rk and

Figure 5.30: An example of a function performance report.

Figure 5.30 gives an example of a function analysis report generated in this case. It also provides a set of GPS matrices and function maps for users to adopt and refer to when making decisions on choosing surface texture parameters and their corresponding limited values (see Appendix *F-J*). Moreover, the Function component also provides an interface for users to add new cases based on the pattern language defined in this thesis, see Figure 5.31.


Inputing New Case

Pattern 3 (Surface Parameter Selection)

Context: The desigers need to select the suitable specification for a surface in order to ensure the surface functions correctly

Problem: Determination of the surface parameters to satisfy Pattern 2-Functional Performance of the surface

Solution: There are two basic approaches: 1. Establish Pattern 4 - Functional correlation with texture parameters; 2. First establish a stable surface generation process that produces acceptable surface and then Pattern 5 - Mo

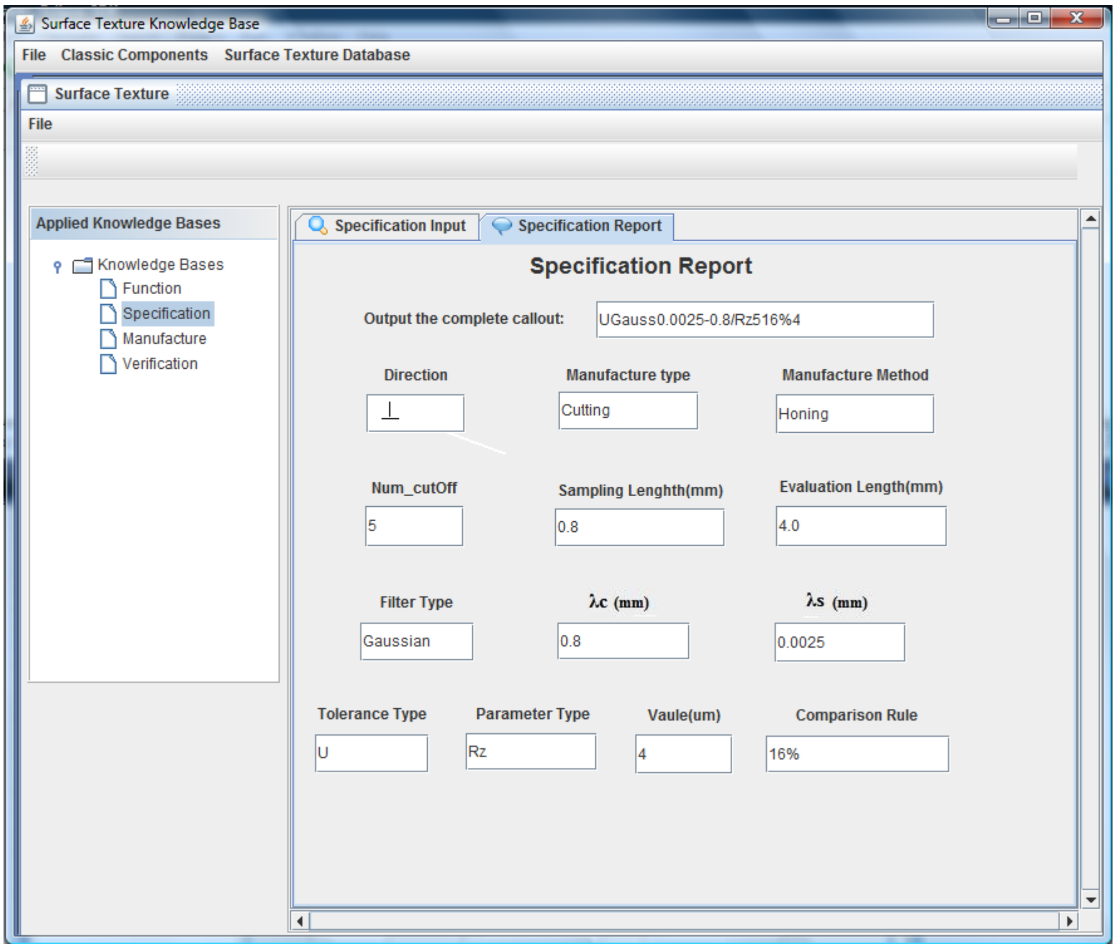
Forces: The functional correlation approach is uperior in quality of results but is more expensive in time and cost to estalish correlation and more sophisticated measuring equipment is required than establishing a stable surface generation process and monitoring fo

Example: In order to find relations between relevant parameters with surface functions of femoral heads, it is better to establish a stable surface generation process that produces acceptable surfaces and then Pattern 5 - Monit

Next Pattern (Pattern 2)

Figure 5.31: The new case inputting interface for users.

Step 4: After retrieving the function analysis report, the next design stage will move on to the Specification component. It provides users the complete '*callout*' on drawing for the specific surface texture, defined by sampling length, evaluation length, bandwidth for the filter, and so on. Figure 5.32 shows the output of this module for the cylinder liner with *Rz* defined at 4 μm.

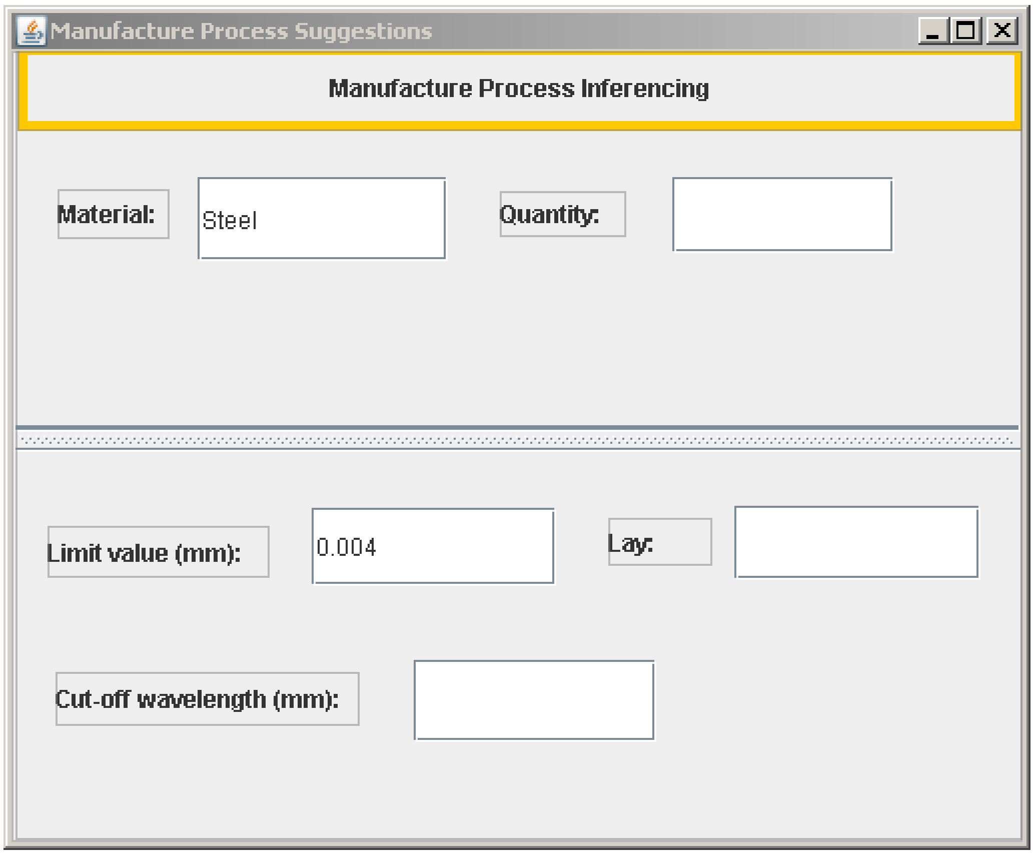


Specification Report

Output the complete callout:	UGauss0.0025-0.8/Rz516%4		
Direction	Manufacture type	Manufacture Method	
⊥	Cutting	Honing	
Num_cutOff	Sampling Lenghth(mm)	Evaluation Length(mm)	
5	0.8	4.0	
Filter Type	λc (mm)	λs (mm)	
Gaussian	0.8	0.0025	
Tolerance Type	Parameter Type	Vaule(um)	Comparison Rule
U	Rz	4	16%

Figure 5.32: Output of a specification report.

Step 5: After acquiring the detailed specification report, designers will enter the Manufacture component of the VirtualGPS system. The Manufacture component searches for appropriate manufacturing processes for users. Based on material types and quantity entered by designers, as well as texture lay and limit values calculated by the specification report, this component infers suitable manufacturing processes among several GPS matrices (e.g. manufacturing process PRIMA selection matrix) using a set of in-built inference rules. Figure 5.33 is an interface to allow users to input inference properties for the Manufacture component.


Manufacture Process Suggestions

Manufacture Process Inferencing

Material: Steel

Quantity:

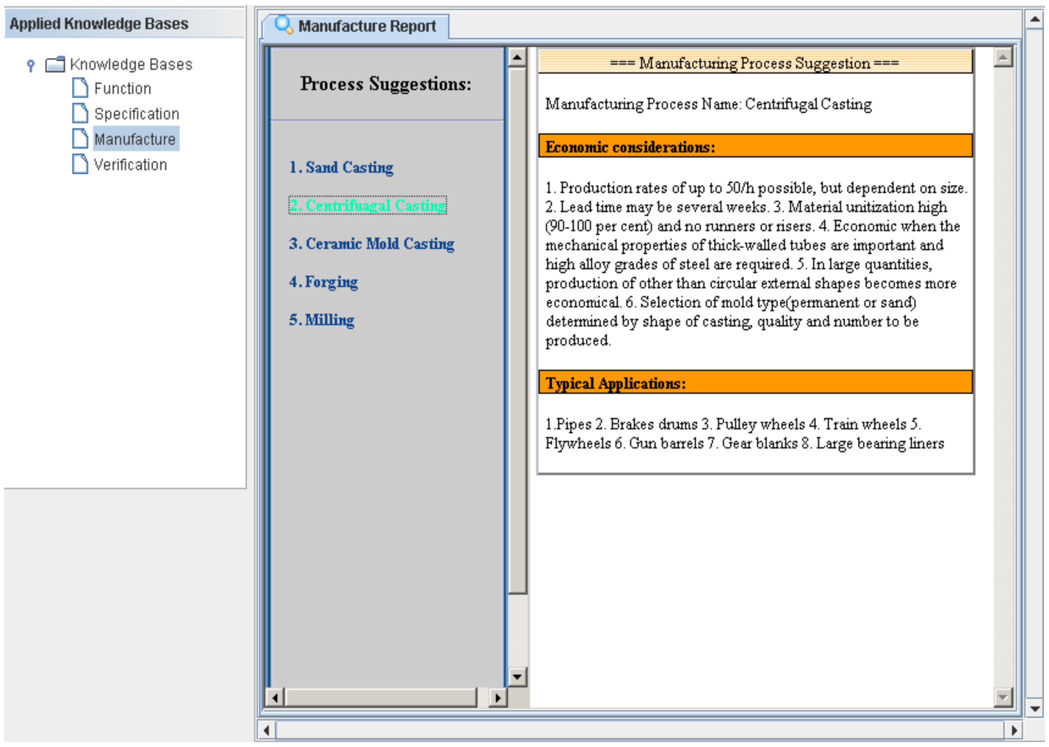
Limit value (mm): 0.004

Lay:

Cut-off wavelength (mm):

Figure 5.33: Inputs of the Manufacture component.

In this case, after inputting material with the value of “steel” and limit value (tolerance value) of 0.004mm, the Manufacture component can generate a manufacturing process report for guiding users to choose suitable manufacturing processes, see Figure 5.34.



Process Suggestions:

1. Sand Casting

2. Centrifuagal Casting

3. Ceramic Mold Casting

4. Forging

5. Milling

=== Manufacturing Process Suggestion ===

Manufacturing Process Name: Centrifugal Casting

Economic considerations:

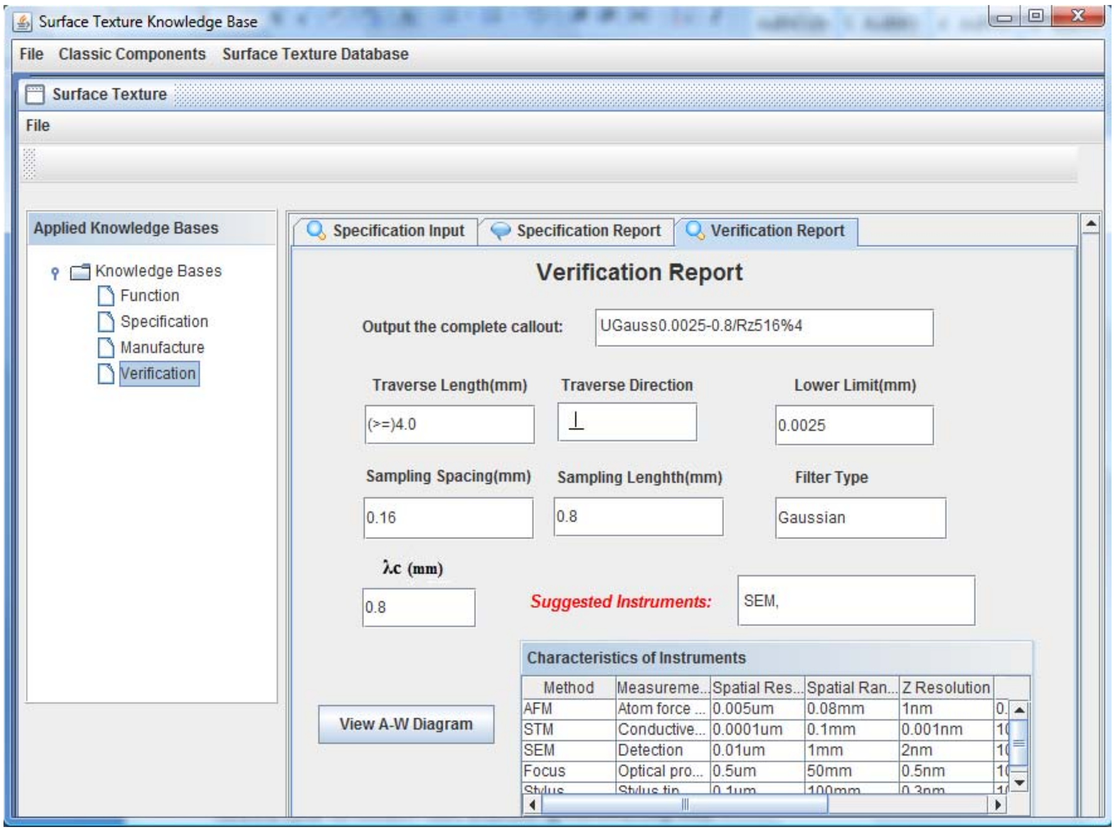
1. Production rates of up to 50/h possible, but dependent on size. 2. Lead time may be several weeks. 3. Material unitization high (90-100 per cent) and no runners or risers. 4. Economic when the mechanical properties of thick-walled tubes are important and high alloy grades of steel are required. 5. In large quantities, production of other than circular external shapes becomes more economical. 6. Selection of mold type(permanent or sand) determined by shape of casting, quality and number to be produced.

Typical Applications:

1.Pipes 2. Brakes drums 3. Pulley wheels 4. Train wheels 5. Flywheels 6. Gun barrels 7. Gear blanks 8. Large bearing liners

Figure 5.34: Output of a manufacture report.

Step 6: The final design step uses the Verification component to find out suitable measurement information for the cylinder liner, which can include traverse length, sampling space, measuring instruments. Figure 5.35 shows the interface of the generated verification report.



Verification Report

Output the complete callout: UGauss0.0025-0.8/Rz516%4

Traverse Length(mm): (>=)4.0

Traverse Direction: ⊥

Lower Limit(mm): 0.0025

Sampling Spacing(mm): 0.16

Sampling Lenghth(mm): 0.8

Filter Type: Gaussian

λc (mm): 0.8

Suggested Instruments: SEM,

Characteristics of Instruments

Method	Measureme...	Spatial Res...	Spatial Ran...	Z Resolution	
AFM	Atom force ...	0.005um	0.08mm	1nm	0.
STM	Conductive...	0.0001um	0.1mm	0.001nm	1(
SEM	Detection	0.01um	1mm	2nm	1(
Focus	Optical pro...	0.5um	50mm	0.5nm	1(
Stylus	Stylus tip	0.1um	100mm	0.3nm	1(

Figure 5.35: Output of a verification report.

In the verification phase, valid suggestions of accurate measurement instruments are very important. In this project, the so-called 'A-W' diagram is used to make this decision. As shown in Figure 5.36, the small triangle ("Δ") symbol in the figure represents an 'A-W' plot for parameter *Rz* 4 μm with sampling spacing 0.16mm. The user interface can also allow users to zoom in and out of the diagram to check details of the coordinate plot information.

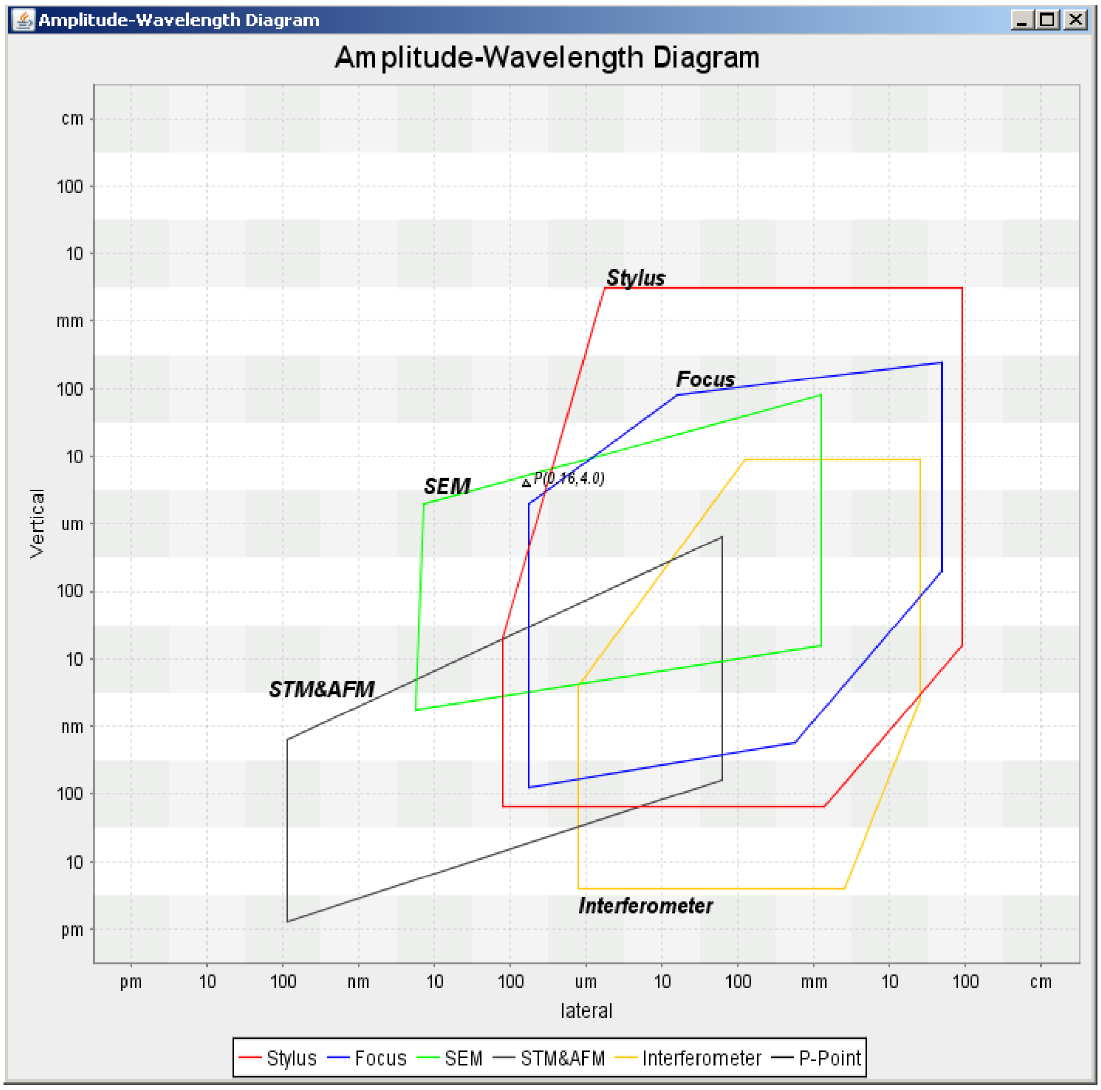
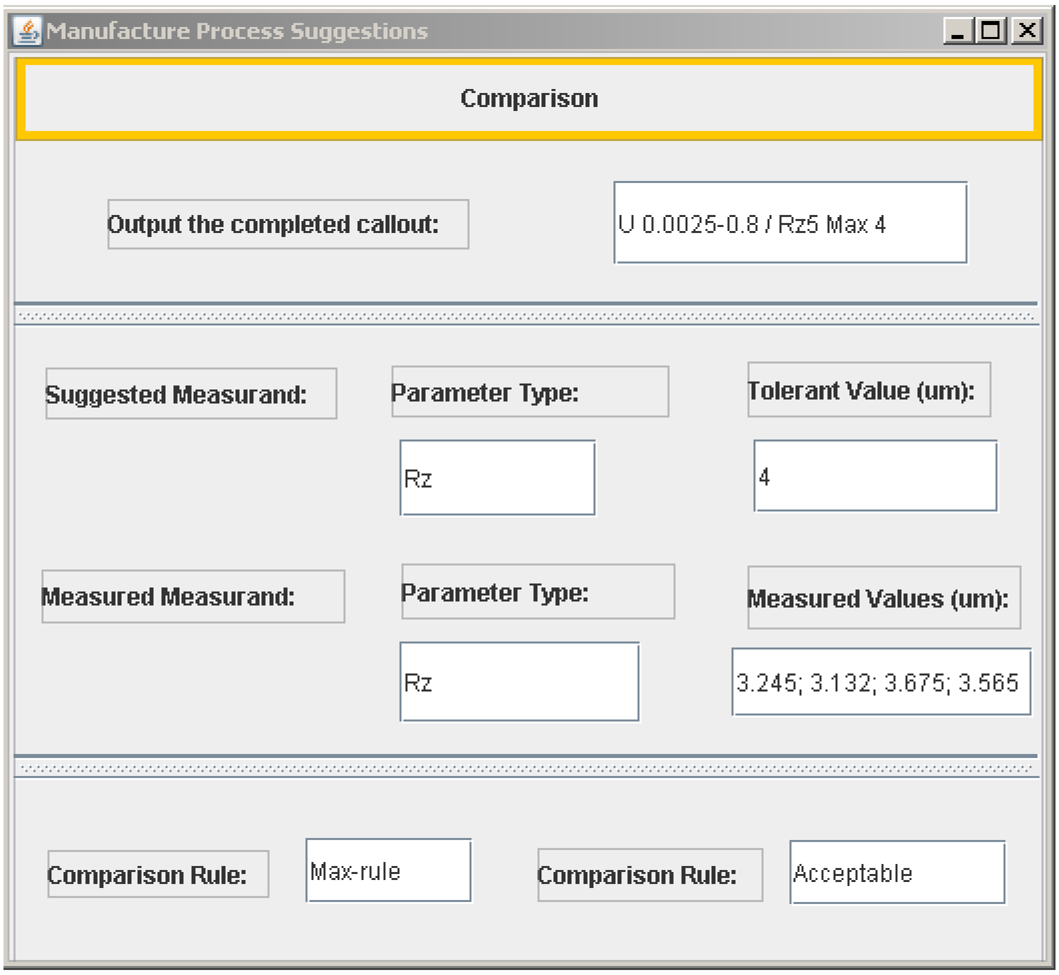


Figure 5.36: Visual representation of an A-W diagram.

The Figure 5.37 shows an output of a comparison process.


Manufacture Process Suggestions

Comparison

Output the completed callout: U 0.0025-0.8 / Rz5 Max 4

Suggested Measurand:

Parameter Type: Rz

Tolerant Value (um): 4

Measured Measurand:

Parameter Type: Rz

Measured Values (um): 3.245; 3.132; 3.675; 3.565

Comparison Rule: Max-rule

Comparison Rule: Acceptable

Figure 5.37: Output of a comparison result.

5.7 Summary

The entire VirtualGPS system is designed and implemented conforming to the Category Theory and the object-oriented programming rules. After the initial tests and analysis performance, it is evident that the system can facilitate the entire geometric product manufacturing lifecycle and benefit the manufacturers and engineers alike from function designs to manufacture and verification. Future work of this project aims at adding more task specific features to help GPS users to improve the design and manufacture geometrical products. A fuzzy logic-based inference engine has also been planned to improve the "intelligence" of the VirtualGPS.

CHAPTER 6 TESTS AND EVALUATIONS

This chapter records in detail the tests and evaluations of the VirtualGPS system, which contains two main parts — categorical DBMS evaluation and the host system evaluation. The evaluations of such a categorical DBMS in this project were carried through comparing with other classic relational, object-relational and object-oriented DBMSs. Qualitative analysis was performed based on several selected evaluation cases.

6.1 Tests and Evaluations on the Categorical DBMS

In this section, selected test cases will be used to demonstrate and assess the categorical DBMS through comparison with other types of DBMSs. Based on the test results, critical evaluations will be performed to analyse the pros – and – cons of the categorical DBMS for the VirtualGPS system.

6.1.1 Data Model Comparisons

As the categorical DBMS developed in this research was based on the categorical object model, the first test at the evaluation stage had been focused on the comparison in between the categorical object model and the other two main stream data models widely used at present: the relational data model and the ODMG object model. Currently, there are around 40 commercial relational DBMS products developed by various vendors (e.g. Oracle, SQLServer and MySQL), which have been the dominating force in the database market for the last three decades. One of key factors contributing to the success of relational DBMSs is that they all share a formal and stable basis – the relational data model based on the Set Theory in mathematics. On the other hand, the current ODMG standard 3.0 adopted by most mainstream object-oriented DBMSs such as Objectivity, Versant and ObjectStore, had suffered from the lack of a rigid mathematical definition and practical abilities in dealing with new and innovative data forms (Cattell et al., 2000 [20]). Table 6.1 demonstrates a comparison of these three data models in respect of their modelling capabilities and mathematical supports.

	Relational data Model	**ODMG Object Model**	**Categorical Object Model**
Modelling Capability			
Formal relationship structure (including n-ary)	YES (Based on the Descartes in Set Theory)	NO	YES (Based on the product construct in Category Theory)
Trees/Collections/Arrays	NO	YES	YES
Inheritance	NO	YES	YES
Aggregation	NO	YES	YES
Multi-level mappings	NO	NO	YES
Object nests	NO	YES	YES
Mathematical Support			
Manipulations	YES (Based on set operations, algebra and calculus)	NO	YES (based on arrow mapping, arrow composition and functor composition)
Methods/Dynamic Constraints	NO	YES (Based on Object Definition Language without mathematical support)	YES (Based on method arrows)
Normalization	YES (Based on functional dependency checking on sets)	NO	YES (Based on arrow composition checking on categories)
Referential Integrity	YES (Based on foreign key definitions)	YES (Based on object identifiers)	YES (Based on initial internal objects of categories)
Membership/cardinality	YES (by labels)	NO	YES (by typing functors)

Table 6.1: Comparison of three data models.

As highlighted in Table 6.1, the key features for the relational data model can be summarised as: a structure with a sound mathematical foundation that supports a clear and formal construct ("table") for data modelling and it also provides a rigorous data manipulation mechanism based on the relational algebra and calculus on sets. However, it is relatively weak in modelling of complex object structures, especially when modelling multi-level constraints/mappings and object nests architecture. On the other hand, the key features for ODMG object model can be summarized as: the

ODMG object model has strong capability for modelling complex object structures, but lack of a formal mathematical foundation. So it is difficult to ensure the integrity and consistency of a database schema in an object model driven database when manipulations such as deletion, updating and adding occurred, which had been observed by database developers when designing "pure" object-oriented DBMSs.

However, Table 6.1 shows the categorical object model can satisfy both objectives well − having sufficient capability for modelling complex object structures, especially in handling the multi-level constraints and mappings, while offering a rigorous mathematical foundation based on the Category Theory, similar to the Set Theory in a relational data model. The categorical object model provides a uniform way to model both static (attributes) and dynamic (methods) aspects of an object by using different types of arrows. In addition, it defines a manipulation language based on the functor mappings and compositions to ensure integrity and consistency of a database schema through diagram chase and algebraic deduction. This is the rationale for devising an object-oriented DBMS based on the categorical object model to provide a stable and powerful foundation for the virtualGPS system.

6.1.2 Test Case 1: Comparing with a Relational DBMS

As stated in Section 2.2.2.4 of Chapter 2, relational DBMSs in general are relatively weak in dealing with many-to-many relationships and other complex nested and embedded structures. As a common practice, dynamic data structures such as lists, collections or other linked data structures are avoided in relational DBMSs. In this section, two examples derived from this project will be used to highlight why relational DBMSs were not adopted in this project. A classic relational DBMS − MySQL was chosen for this analysis.

6.1.2.1 Operations on Object Nests

This test was based on a simple object nest example to show the basic differences between the categorical DBMS and relational DBMSs. Table 5.1 in Chapter 5 has provided guidance to link the surface requirements with functional performances. For example, if the surface requirements are two solid bodies in contact with a rolling motion between them, then the most important functional demands for the surface is the wear fatigue. As almost all relational DBMSs do not support object nests (table embedded), the data in Table 5.1 must be separated into several small tables (BCNF obeyed) and to be "glued" using foreign keys, which can be interpreted into MySQL in following form:

```
CREATE TABLE MECHANISMOFWEAR (Wear_ID char(5) NOT NULL, Name char(20),
Important_Level char(20), Primary Key (Wear_ID));
CREATE TABLE RELATIVEMOTIONS (Motion_ID integer NOT NULL, Type char(10),
Scheme blob NOT NULL, Primary Key(Motion_ID));
CREATE TABLE SURTOFUN (Element char (20) NOT NULL, Motion_ID integer,
Type_Of_Wear char(20), Mechanism_of_wear char(5), Primary Key(Element), Foreign Key
(Mechanism_of_wear) references MECHANISMOFWEAR (Wear_ID) AND (Motion_ID)
references RELATIVEMOTIONS (Motion_ID));
```

List 6.1: SQL code list for creating linked tables.

Based on List 6.1 above, a query can be formed as “Print the most important surface functional demands for two solid bodies contacting with a sliding motion between them”:

```
SELECT Element, RELATIVEMOTIONS.Type Relative_motions_type,
 RELATIVEMOTIONS.Scheme Relative_motions_scheme,
Type_of_wear, MECHANISMOFWEAR.Name mechanism_of_wear_name,
MECHANISMOFWEAR.Important_level mechanism_of_wear_importantLevel
FROM SURTOFUN, RELATIVEMOTIONS, MECHANISMOFWEAR
WHERE ELement = “Solid body/Solid body” AND Motion_ID = {
        SELECT Motion_ID FROM RELATIVEMOTIONS
         WHERE Type = “Sliding”
 }
AND Mechanism_of_wear = {
        SELECT Wear_ID FROM MECHANISMOFWEAR
        WHERE Important_Level = “Most important”
};
```

List 6.2: SQL code list for querying linked tables.

The results for this query operation can be displayed on screen by MySQL as Table 6.2 shown:

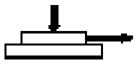
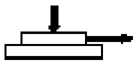
Element	**Relative _motions_ type**	**Relative_ motions_scheme**	**Type_of_wear**	**mechanism_of _wear_name**	**mechanism_of _wear_importantLevel**
Solid body Solid body	Sliding		Sliding wear	Adhesion	Most important
Solid body Solid body	Sliding		Sliding wear	Tribocorrosion	Most important

Table 6.2: MySQL query results.

In contrast, due to the object nest supporting, the Figure 6.1 demonstrates a categorical modelling of Table 5.1, which can be directly stored in the categorical DBMS.

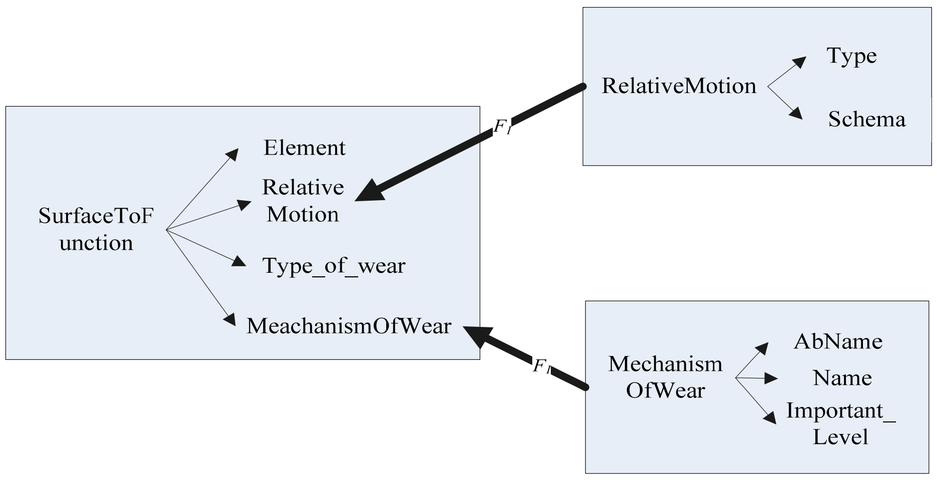


Figure 6.1: A categorical object model for the linkage between surface requirements and functional performances.

F_1 and F_2 are "faithful" functors which injects subclass categories — "*RelativeMotion*" and "*MechanismOfWear*" into a superclass category "*SurfaceToFunction*" while preserving their structures. The Figure 6.1 actually represents a tree structure that indicates a high-level aggregate category containing two lower-level sub-categories. Therefore, the class category "*SurfaceToFunction*" can be formed as in List 6.3:

```
public class SurfaceToFunction {
    public Arrow interObjId_id;
    public Arrow interObjId_Element;
    public Arrow interObjId_RelativeMotion;
    public Arrow interObjId_TypeOfWear;
    public Arrow interObjId_MechanismOfWear;

     public void setArrows(Arrow interObjId_id, Arrow
                               interObjId_RelativeMotion, Arrow
                               interObjId_TypeOfWear, Arrow
                               interObjId_MechanismOfWear){
        this.interObjId_id = interObjId_id;
        this.interObjId_Element = interObjId_Element;
        this.interObjId_RelativeMotion = interObjId_RelativeMotion;
        this.interObjId_TypeOfWear = interObjId_TypeOfWear;
       this.interObjId_MechanismOfWear =
                                                interObjId_MechanismOfWear;
    }

    public void setTargetForIdArrow(int id){
          this.interObjId_id.setTarget(Integer.valueOf(id));
   }

   public void setTargetForRelativeMotionArrow(RelativeMotion
                                                     relativeMotion){
         this. interObjId_RelativeMotion.setTarget(relativeMotion);
   }

   public void setTargetForMechanismOfWearArrow(MechanismOfWear
                                                        mechanismOfWear){
```

```
            this. interObjId_MechanismOfWear.setTarget(mechanismOfWear);
        }
        …………
        //set and get methods for arrows
    }
```

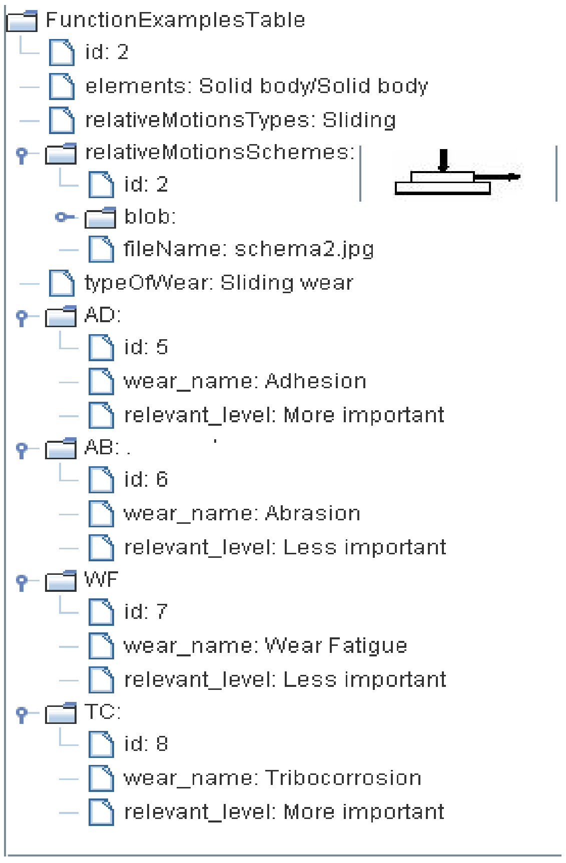
List 6.3: Code list for "*SurfaceToFunction*" class category.

The class categories "*RelativeMotion*" and "*MechanismOfWear*" can be defined in same way as the "*SurfaceToFunction*". Therefore, a query can be formed as in List 6.4.

```
query.constrain(RelativeMotion.class);
query.descend("interObjId_Element").descend("target").constrain("Solid body/Solid body
").and(query.descend("interObjId_RelativeMotion
").descend("target").descend("interObjId_Type").descend("target").constrain("Slibing").and
(query.descend("interObjId_MechanismOfWear
").descend("target").descend("interObjId_ImportantLevel").descend("target").constrain("mo
st important");
```

List 6.4: Categorical query codes for the linkage between surface requirements and functional performances.

The results are then displayed by the categorical DBMS as a hierarchical tree structure as Figure 6.2 demonstrated.



```
FunctionExamplesTable
  id: 2
  elements: Solid body/Solid body
  relativeMotionsTypes: Sliding
  relativeMotionsSchemes:
    id: 2
    blob:
    fileName: schema2.jpg
  typeOfWear: Sliding wear
  AD:
    id: 5
    wear_name: Adhesion
    relevant_level: More important
  AB: .
    id: 6
    wear_name: Abrasion
    relevant_level: Less important
  WF
    id: 7
    wear_name: Wear Fatigue
    relevant_level: Less important
  TC:
    id: 8
    wear_name: Tribocorrosion
    relevant_level: More important
```

Figure 6.2: The query result in tree structure.

This innovative form for generating and displaying query results has enabled potential applications such as enable faster and safer database queries, prevent data corruption, reduce table joins and provide simple integrity checking. Some important differences between the relational DBMSs and the categorical DBMS can be summarized as shown in Appendix *K*.

6.1.2.2 Test on the Comparison Processes in Verification

In the verification step, the VirtualGPS system users can verify the measured values of a product in accordance with tolerance values of GPS parameters suggested by the Specification component of the system. In order to support this function, the DBMS should have the ability to store the measurands, measured values, comparison related information and comparison results for further queries. A test case for testing this ability is defined at here: the surface texture knowledge base of the VirtualGPS system suggests that the measurand for a cylinder liner is the surface parameter *Rz* with a tolerance value of 4 μm. Table 6.3 lists the measured values of *Rz* calculated on a manufactured cylinder liner.

Cylinder liner	**Rz (μm)**
No.1	3.245
No.2	3.132
No.3	3.675
No.4	3.565
No.5	3.175
No.6	3.432

Table 6.3: Surface parameter *Rz* calculated on a manufactured cylinder liner.

The comparison information contains the comparison rule − "max-rule" (where the requirements specify a maximum value of the parameter, none of the measured values of the parameter over the entire surface can exceed the suggested tolerance value.), the measurement instrument (revolution, space), and the comparison result etc. By using the inference identifying square illustrated in Figure 3.22 and as well as the corresponding categorical sequence diagram in Figure 3.18, the knowledge generated from the comparison processes can be directly stored in the categocial DBMS. In Figure 3.22, F_1 and F_2 are functors mapping from the category "*MeasurandForComparison*" to the category "*Value*". The σ is a natural transformation mapping from F_1 to F_2. The F_1, F_2 and σ form a natural transformation square in the form explained in Figure 3.4 of Chapter 3. The natural transformation σ should keep the diagram commuting as defined in the Category Theory, which means that two paths drawn from the values for domains of arrows in the source category

$F_1(dom(f_i))$ to the values for the codomains of arrows in the target category $F_2(cod(f_i))$ should be equal. In this case, a natural transformation square is used to link the suggested measurement pairs (from the GPS surface texture parameters to the tolerance values suggested by the specification part) to the measured pairs (from the measurands to the measured values) inputted by users. As mentioned earlier, Figure 3.22 also contains a 2-ary product relationship structure between the natural transformation square and a class category "*Comparison*". The "*ComparisonResult*" is a class category for storing all information generated from this relationship link (e.g. comparison result, resolution of measurement instrument, traverse range of measurement instrument, meansurands, and measured value). Keeping these multi-level mappings intact in the database is very important, because it is useless to store only comparison results for verification without knowing the corresponding suggested measurement pairs and measured pairs. Using the instance categories of the "*MeasurandForComparison*", "*Value*", "*Functor*", "*NaturalTransformation*" and "*Comparison*" as input of the verification inference rules (e.g. *max-rule*) in the VirtualGPS, the final comparison result together with related comparison information will be stored in the instance categories of the "*ComparisonResult*". All arrow mappings, functor mappings, will be preserved and all constraints (e.g. the parameter type in source side of natural transformation σ must equal to target side) will be checked.

To implement this case in a relational DBMS, the first problem is that it is impossible to store dynamic data structures in relational DBMSs. The data structures that are dynamic indicate their data size can grow and shrink while computing programs are running. Table 6.4 shows the performance differences between static and dynamic data structures.

	Static Data Structures	**Dynamic Data Structures**
Data Size	Size is fixed when declared	Size is not fixed
Storage efficiency	Inefficient storage due to oversizing (e.g. a partially full array, but space has been allocated for the full size)	Efficient storage(e.g. space can be allocated as a partially full linked list required)
Flexibility of update	Inflexible(e.g. if one more value needs to be added overrunning the maximum size, the array needs to be redeclared and populated)	Flexible(If one more value needs to be added overrunning the maximum size, the linked list increases automatically)
Execution speed	Fast at execution	Slow at execution

Table 6.4: Static vs. dynamic data structures.

It became obvious at the system design phase that dynamic data structures are much more suitable for holding data at runtime due to the unknown size of the measured values in advance. Hence, the ability for storing and retrieving dynamic data structures is an important feature for the system implementation. In the devised categorical DBMS, the class category "*Value*" that extends a dynamic data structure "*CTTree*" can be used to store an arbitrary size of measured values.

The second problem is that the relational DBMSs are incapable of recording the natural transformation mappings in a traceable manner since they do not conform to the normalization rules, which will cause loss of constraints during a persistence test.

6.1.2.3 Result Analysis

Based on the experiments explained above, several advantages of the devised categorical DBMS over conventional relational DBMSs can be summarized as:

- Uniform mapping from design to implementation. In the relational design stage, database developers model business applications in the form of E-R diagrams. Then when implemented, developers need to translate the E-R diagrams into "Table" based database schema based on Normal Form, primary key definitions and foreign key linkages, which is a time consuming and error prone process. In the categorical process, database developers only need to model business applications in the categorical object model forms and then directly store them in the categorical DBMS.
- Novel support for the object nest (category nest) and multi-level mapping structures. Categorical DBMSs can directly store and retrieve nested objects without any extra hierarchical definitions on links. It also provides a visual tree structure to facilitate the display of nested objects (as shown in Figure 6.2), which is much clearer than table based results displayed in Table 6.2.
- Supporting storage and retrieval of dynamic data structures. Also, categorical DBMS can store and retrieve dynamic objects computed through a method.
- Simple query strategy with robust query closure. Queries from relational DBMSs are often cumbersome since the PSJ operations (Project, Select, Join) have to be called frequently to reconstruct objects. The categorical DBMS has avoided this drawback through implementing a more natural and robust querying mechanism.

It has been observed during the aforementioned experiments, due to the clear and logical mappings between applications and databases; natural representations of data structures and database results in fewer codes; and Java based garbage collector, the devised categorical DBMS is in average 10 times faster than an analogical mySQL product when processing a query operation. The system is also capable of updating or deleting an object nested more than 10 levels in any object hierarchical tree, as well as average 1/3 memory cost of traditional relational DBMSs when contain more than 500k data in memory.

6.1.3 Test Case 2: Comparing with a Object-relational DBMS

This section examines the performance differences between the categorical DBMS and an object-relational system — P/FDM, a research development by the Object Database Group at the University of Aberdeen (Embury, 1995 [30]). The P/FDM is based on a functional data model using a hybrid DAPLEX and SICStus prolog query interface (Intelligent Systems Laboratory, 2006 [31]). The functional data model was formed by entities and functions mapping from entities to other entities. Both entities and functions are become tables in the P/FDM's physical level. The P/FDM contains three object-oriented extensions:

- Entity nest
- Entity Inheritance
- Function can either be persistent relations or derived methods

For storing functions in the database, the so-called "function table" in P/FDM was used, which can support none atomic columns (Table nests). To compare with the devised categorical DBMS, an example of using the P/FDM to implement the callout schema defined in the Specification module of the VirtualGPS had been carried out. The database schema definitions for the complete callout in the P/FDM are expressed in Appendix *L*.

Figure 6.3 graphically shows this database schema in the functional data model form, which is an extension of E-R diagram for modelling object-relational DBMSs.

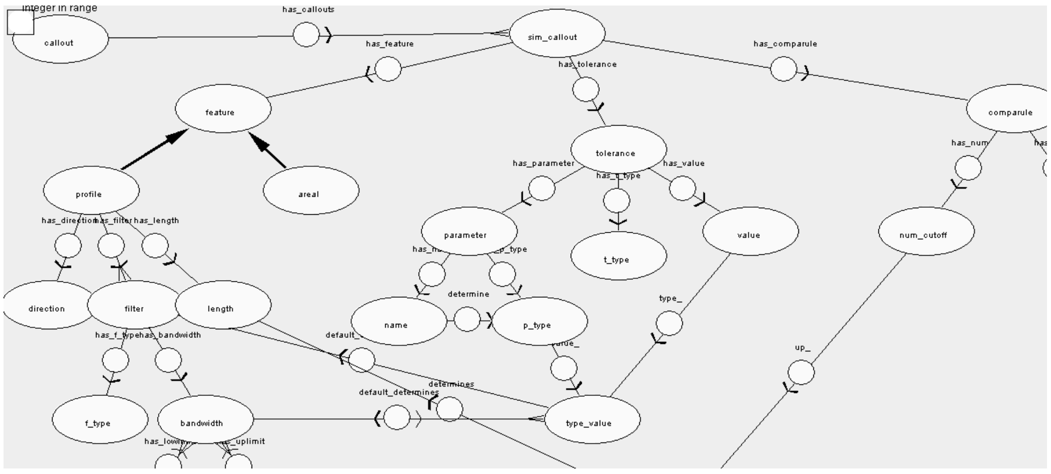


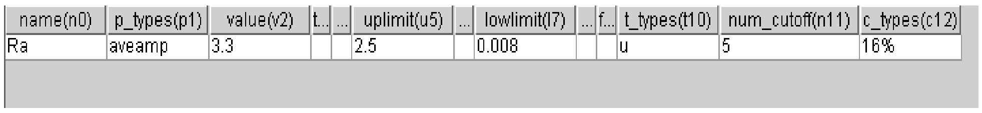
Figure 6.3: The functional data model for callouts generated by the P/FDM.

After defining the entities and functions, the database can be populated with real data. Some snippet codes for populating callouts in P/FDM can be found in Appendix *M*. Appendix *M* contains a set of nested tables for realizing functions (e.g. function "*default_determines*") and relationships (e.g. "*bandwidth*"). The query clause "Print the default semi-completed callout symbols for '*Ra* 3.3' was then generated (without manufacture methods, direction and machine allowance)" in the internal P/FDM form as List 6.5:

```
for each n0 in name such that name(n0)="Ra"
for each p1 in p_type such that p1=determine(n0,"p")
for each v2 in value such that value(v2)=3.3
for each t3 in type_value such that types(t3)=p_type(p1) and range(t3)=range(v2)
for each b4 in bandwidth such that b4=default_determines(t3)
for each u5 in uplimit such that u5=has_uplimit(b4)
for each b6 in bandwidth such that b6=b4
for each l7 in lowlimit such that l7=has_lowlimit(b6)
for each d8 in direction such that d_name(d8)=""
for each f9 in f_type such that f_name(f9)=""
for each t10 in t_type such that t_name(t10)=""
for each n11 in num_cutoff such that n_name(n11)=""
for each c12 in c_type such that c_name(c12)=""
print(name(n0),p_type(p1),value(v2)," "," ",uplimit(u5),"
",lowlimit(l7),direction(d8),f_type(f9),t_type(t10),num(n11),c_type(c12));
```

List 6.5: P/FDM query clause for the callouts.

The query results are shown in Figure 6.4:



name(n0)	p_types(p1)	value(v2)	t...	...	uplimit(u5)	...	lowlimit(l7)	...	f...	t_types(t10)	num_cutoff(n11)	c_types(c12)
Ra	aveamp	3.3			2.5		0.008			u	5	16%

Figure 6.4: Output for the semi-completed callout symbols from the P/FDM.

The P/FDM DBMS also defines constraints for enabling integrity checks. For example:

```
constrain each v in Value  to have value(v) <10.0
```

List 6.6: Example for a constraint definition in the P/FDM.

Thus, values input by users for the "*Value*" entity can only be floating values less then 10.0. Further features of the P/FDM DBMS can be summarized:

- All database entities are stored in tables with some of the tabular entries may have richer data structure ─ Abstract Data Types (ADTs).
- Supports constraint definition.
- Supports user defined data types and complex object query.
- Query results of P/FDM are also table or tuple (row) formats, so query results can be stored back to the database to form further queries.

The database schema definitions and populations for callouts in the categorical DBMS have been explained in Section 4.4.1. After instances of "*Callout*" class category have been populated by the initial query in the Section 4.4.1.5, the direct query clause for "Print the completed callout symbols for '*Ra* 3.3' (without manufacture methods, direction and machine allowance)" in categorical DBMS can be generated as List 6.7:

```
query.constrain(Callout.class);
query.descend("interObjId_measurand_paraType").descend("target").constrain("Ra").and(q
uery.descend("interObjId_limitedValue").descend("target").constrain(3.3));
```

List 6.7: A direct query for callouts in the categorical DBMS.

In summary of above discussions, the devised categorical DBMS has four main advantages over conventional object-relational DBMSs:

- Although object-relational DBMSs and the categorical DBMS can both directly map data models into implementations, the data models for object-relational DBMSs are still weak in terms of semantic supports. For example, the functional data model in the P/FDM does not have a structure like the "product" as in the categorical object model. Therefore, Figure 6.3 can only tell database programmers that the database entity "*Callout*" is a relationship entity generated by linking the "*Feature*", "*Toleranc*e" and "*Comparule*" without indicating what the real information should be held in the database entity "*Callou*t". This also leads to the situation where the P/FDM can not support direct query on the "*Callout*" entity as the categorical DBMS did in this section. In the categorical DBMS, with the product construct, the relationship category "*Callout*" can be defined clearly in advance with all the essential information in respect of constraints, so an initial query (see Section 4.4.1.5) can be devised to populate instances for "*Callout*", and then the query results can be stored back for a direct query on the "*Callout*" as illustrated in this section.

- By Comparing Figure 3.30 and Figure 6.3, it is clear that the callout schema modelled in the categorical object model is much clearer and simpler than that is in a functional data model. The functional data model extends the relational data model with some object-oriented features such as inheritance and entity nests, but is largely behavioring in a relational manner, for instance:
  1) Objects need to be identified by primary keys. Foreign keys are used to link with other entities. Moreover, the entity nests are achieved in form of key nests.
  2) Relationship functions are used to define relations, which supports many to many relationships and entity nests. However, they are also based on key or key nests.
- Although object-relational DBMSs such as the P/FDM support method/behaviours, they are incapable of supporting dynamic methods. The so-called "method" in the P/FDM still carries heavy overhead. For example, database application developers need to populate every function with all its possible inputs and all possible outputs in advance (e.g. function "*default_determines*"), which is a heavy overhead and error prone process. In addition, in order to obtain the default roughness sampling length based on the recommended surface parameters, the inference rules explained in table 5.6, 5.7, 5.8, 5.9 and 5.10 of Chapter 5 will need to be applied. For keeping sound encapsulation of application objects and simplifying business logics in an application, the best solution is to define those rules as dynamic methods in corresponding application objects, which is beyond reach of the P/FDM and other object-relational DBMSs.
- As suffered by other object-relational DBMSs, the P/FDM is also weak in dealing with relationship or constraints crossing multiple levels, which only in favour of the flat functions in a single level as defined in the Set Theory.

6.1.4 Comparing with an Object-oriented DBMS

The main problem for other "pure" object-oriented DBMSs, such as DB4O, in implementation of the test case discussed in Section 6.1.2.2 is that they can not support the categorical object model directly. Different database application developers could end with totally different approaches to define classes, which cause great difficulties for code reusability and modularized design. Misunderstandings can easily occur in between GPS knowledge base designers and the database application developers, as objects in a database are very different from objects in an actual application. Due to the absence of the multi-level mapping constructs in most of the conventional object-oriented DBMSs, the multi-level relationships and constraints will be largely missed out during persistence tests. A classic usage of object-oriented DBMSs is to directly merge one class into another class to form a relationship between these two classes, which causes the following problems:

- Lack of a rigorous class definition for holding the information generated from the relationship linkage. Hence, queries for the relationship information are difficult to form. The query closure also becomes difficult due to the lack of a formal relationship structure.
- It is difficult to check the cardinality and membership for a relationship. This also leads to the unnecessary complexity for updating or deleting objects involved in a relationship from the database.
- The BCNF normalization rule violation.

Table 6.5 gives a summary of performance differences between the devised categorical DBMS and the object-oriented DBMSs

	Categrical DBMS	**Object-Oriented DBMS**
Structures		
Formal relationship structure (including n-ary)	YES	NO
Trees/Collections/Arrays	YES	YES
Inheritance	YES	YES
Aggregation	YES	YES
Multi-level represenation	YES	NO
Rules		
Normalization Support	YES(without atomicity rule of 1NF)	NO
Referential Integrity	YES	NO
Membership	YES(by typing functors)	YES(by labels)
Manipulation		
Algebra/Calculus	YES(based on arrow mapping, arrow composition and functor composition	NO
Declarative Query	YES	YES
Closure	YES	NO
View	YES	NO
Methods	YES	YES

Table 6.5: A Comparison between the categorical DBMS and the object-oriented DBMSs.

Based on Table 6.5, the realized extensions of the devised categorical DBMS and its advantages can be summarized as:

- An innovative categorical object model.
- A distinct mechanism for dealing with multi-level architecture.
- A manipulation language with the intrinsic query closure capability.
- An integrity checking mechanism in both intra and inter category levels.

The detailed discussions on the four points above can be found in Section 7.1.2 of Chapter 7. During the system test and performance evaluation, it was observed that the main shortcoming of the current categorical DBMS is its heavy dependency on the Java language. Although greatly simplified the development cycle, database application developers adopting the categorical DBMS must possess sound knowledge in Java programming. Researcheres in this project try to devise an Object Definition Language (ODL) that is independent of any real programming languages based on the ODMG standard 3.0 to alleviate this shortcoming. This work is ongoing. The categorical DBMS is not intended to support more database concepts than other DBMSs. Rather it aimed at and successfully achieved to provide a formal mathematical basis for modern object-oriented DBMSs. It formed the backbone for fully supporting the design and implementation requirements of the VirtualGPS in Java.

6.2 Evaluation of the VirtualGPS System

During past three decades, various computer aided manufacturing software system have been developed to benefit the broader product ranges, shorter model lifetimes, and the ability to process orders in arbitrary lot sizes in global distributed areas, that are common in modern industry. In general, the major software systems at present have three shortcomings:

- The functionality features such as product function specifications, the suggestions of surface properties, the related verification principles, measuring equipment selections, and the measurement traceability mechanism are often largely ignored in current software systems.
- The current systems rely on ambiguous dimensioning and tolerancing practices based on the nominal model methodology and geometry theory. The powers of GPS standards are not fully applied in these systems.
- The current systems have limited ability to provide documentation and storage

mechanisms. Therefore, it is difficult for users to perform systematic measurement analysis or to store relevant knowledge for further communications.

Hence, the VirtualGPS system can be used to remedy the above functionality shortcomings by using the universal GPS standards and Category Theory.

6.3 Summary

This chapter provided detailed discussions on the tests and evaluations carried out on the categorical DBMS and VirtualGPS system. The categorical DBMS forms the foundation and served as a core module for the VirtualGPS system. The devised prototype in this project has proven the feasibility and advantages of the Category Theory based modelling. Although the devised categorical DBMS is still falling short of a fully-fledged DBMS compared with other commercial DBMSs, it has been clearily demonstrated that the categorical DBMS is capable of storing and managing complex data structures inherited from contemporary GPS standards and is also ideal for providing data consistency when generating database schema. The final part of the evaluation exercises in this project shows that the VirtualGPS system can provide distinctive functional sets in supporting product function specification, surface property description, and verification principle recommendation etc., backed up by formal documentation mechansims.

CHAPTER 7 CONCLUSIONS AND FUTURE WORK

This chapter summarizes the outcomes from the project and highlights the contribution to knowledge in the relevant research domains, which were detailed in previous chapters. Further works on the VirtualGPS are also discussed at the end.

7.1 The Summary of Contributions

7.1.1 A Categorical Modelling Mechanism for Knowledge-based Systems

The first main contribution of this project is the production of an innovative and efficient graph-based categorical modelling mechanism. This categorical modelling mechanism contains three components: a categorical object model, a categorical software design process and an inference identifying square. Based on those, the categorical modelling mechanism has provided a highly unified and abstract modelling approach for handling all aspects relating to a knowledge-based system design. The following sub-sections conclude the each individual component.

7.1.1.1 The Categorical Object Model

The categorical object model was developed to model both the application domain knowledge and the database schemas with six distinctive advantages over other conventional data models:

- The multi-level mappings defined in the Category Theory enabled the categorical object model to handle the multi-level features of knowledge structures and database schemas with ease. For example, the natural transformation square discussed in Section 3.6 is difficult to be represented in the Set Theory.
- The diagrammatical notations of the Category Theory provide designers with a high-level abstraction view on system architectures, knowledge structures and database schemas.
- Different types of arrows (e.g. method arrows, functional dependency arrows and functors) provide a powerful and unified style for natural modelling of both dynamic (methods, operations, inferences) and static (attributes, properties, classes) aspects of the knowledge and database schemas.
- The typing mechanism of the categorical object model allows the assignment of types to all instance categories and arrows, which ensures the consistency and robustness for implementions.

- It has provided a formal refinement mechanism.
- The diagram chase and algebra deducing abilities of the categorical object model ensures the integrity of a knowledge base or a database schema as well as assisting version management.

These six advantages listed above demonstrated that the categorical object model is suitable for acquiring and modelling complex structured knowledge with a unique and powerful set of inference rules to support its operations.

7.1.1.2 The Categorical Software Design Process

A categorical software design process was devised in this project based on the Category Theory. It can facilitate software engineers to design and implement the entire system architecture in a well defined framework. This design process is an incremental and refineable one formed by five stages: the categorical business map design, the categorical analysis model design, the categorical design model design, the categorical sequence diagram design, and the categorical deployment model design. This design process provides a set of standard procedures for engineers to carry out the design and implement tasks for a knowledge-based system: analysing and gathering user requirements; acquiring knowledge from user requirements and other references (e.g. GPS standards for this project); organizing knowledge in the forms of class categories or relationship categories for forming the knowledge base; refining these class and relationship categories to identify inference rules for generating new knowledge; building the sequence diagrams to define category interactions and communications; and deploying these categories on the targeted computing resources to carry out implementations.

7.1.1.3 The Inference identifying Square

The inference identifying square is defined by using the natural transformation and coequalizer constructs of Category Theory. It is used to identify inference rules based on existing knowledge, and to specify how inference properties are interacted with inference rules in detail.

7.1.2 The Categorical DBMS

The second main contribution of this project is the development of a categorical DBMS. This categorical DBMS is developed based on the aforementioned categorical object model. Compared with traditional relational DBMSs, the categorical DBMS has strong capabilities in dealing with complex object structures especially for modelling multi-level mapping constraints. Therefore, dynamic data structures and complex structured knowledge modelled in the knowledge base of VirtualGPS can be directly stored and queried without the need to grogram any mapping codes between the data in the database and the data in the application. Furthermore, all arrow and functor mappings, will be preserved and all constraints (e.g. the parameter type in source side of natural transformation σ must equal to target side in Figure 3.22) will be checked when storing or updating data. On comparison with other conventional object-oriented DBMSs, the advantages of the categorical DBMS are highlighted as follows:

- An innovative categorical object model that can map complex object structures into mathematical formulae in Category Theory. It enables algebra and calculus defined in the Category Theory to be used as a formal and rigorous mathematical foundation for ensuring integrity of database schema.
- The categorical object model is powerful and flexible in representing the multi-level architectures, which allows advanced constraint specifications and good extensibility of database schemas to be realized in an application.
- The algebra and calculus such as arrow composition, arrow mapping, functor composition and functor mapping can be used as the basis for implementing a manipulation language with the intrinsic query closure capability. This solution tackled the problem of the lack of a formal manipulation language faced by current object-oriented DBMSs.
- The categorical DBMS has a robust integrity checking mechanism at both the intra- and inter- category levels. Thus, BCNF normal form and referential integrity can be maintained throughout database schemas.

7.1.3 The VirtualGPS Knowledge-based System

The third main contribution of this project is the development of a prototype for the VirtualGPS system to enable theoritical and practical tests and evaluations. Taking surface texture as an example, the Surface Texture module contains four components (sub-knowledge base): Function, Specification, Manufacture and Verification:

- The Function component can help users to select surface texture parameters with tolerance values according to functional requirements. Currently, this selection is inferred based on cases (e.g. cylinder liner and total hip replacement). It also provides an open and modularized platform for engineers or designers to add their own specific cases complying with a specific pattern language format. This pattern language is also tested against and conforming to the Category Theory (see Section 3.7.1 in Chapter 3).

- The Specification component can provide a complete geometrical specification for any suggested parameters in the Function component. The detailed introduction of each symbol in a complete geometrical specification can also be reviewed using the Specification component.
- The Manufacture component contains a rating and ranking inference engine for locating and retrieving any GPS-recommended manufacturing processes and equipments. The relationships between the manufacturing processes and the surface texture parameters can be analyzed. An inputting interface also provides users an input channel for adding new manufacturing processes in the PRIMA forms.
- The Verification component contains an inference engine for determining the verification procedures: selecting an appropriate measuring instrument for determining how to obtain the features from a real surface; suggesting algorithms to calculate the measured parameter values; and comparing the measured values with the recommended tolerance values.

Every component disscussed above can generate a XML report for communications and archiving. This system can also be customized into a piece of training software for helping users to understand and apply the GPS standards in their daily working activities.

7.2 Future Works

Based on project reviews and system evaluations detailed in Chapter 6, some future works to the VirtualGPS system are listed below:

(1) Applying more comprehensively advanced notations and constructs defined in the Category Theory to elaborate the categorical modelling mechanism devised in this project. For example, the advanced diagram injection can be used in the category refinement operations and the ‘monads’ can be use as states for I/O systems (Gordon and Hammond, 1995 [113]).

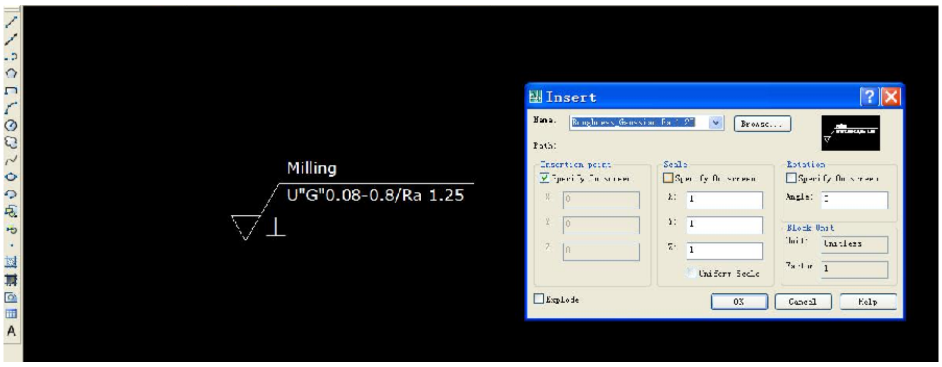
(2) Adding more domain knowledge into the VirtualGPS system. This mainly includes works concerning three aspects:

   - To incorporate more cases into the Function component.

- To incorporate more matrices into the Manufacture component for inferring manufacturing processes based on specifications and other inference properties such as quantity and material.
- To incorporate knowledge for the calibration and uncertainties into the Verification component to allow the generation of a complete measurement process with more measurement instruments.

These goals can be achieved by: continuously enriching GPS standards; continuously acquiring knowledge from experts' publications; continuously gathering new knowledge from virtualGPS system users. As stated in previous Chapters, the current VirtulGPS has limited capability for inferring new knowledge based only on existing cases and defined rules. However, once equipped with enough GPS knowledge, it will be able to reason broadly over the entire field of GPS through applying more advanced inference engines based on fuzzy logic.

(3) Another major development anticipated for the system is to build a portal interface to directly hook this system to other Computer Aided Design (CAD) systems. Thus, the knowledge stored in the VirtualGPS system can be transferred and applied in forming technical drawing pictures automatically. For example, the complete callout can be automatically drawn in an AutoCAD as illustrated in Figure 7.1 demonstrated.



Insert

Name: Roughness_Generation_Ra 1.25
Path:

Insertion point: [✓ Specify On-screen] X: 0 Y: 0 Z: 0

Scale: ☐ Specify On-screen X: 1 Y: 1 Z: 1 Uniform Scale

Rotation: ☐ Specify On-screen Angle: 0

Block Unit: Unit: Unitless Factor: 1

☐ Explode

Figure 7.1: The callout on technical drawing in the autoCAD.

7.3 Finally

In short, it is envisaged that the research and development outcomes from this project will contribute the wider and better adoption of current GPS standards. It is also hoped that the VirtualGPS system will be developed further to handle more complex GPS knowledge inferences to link closer with real world manufactures alike.

RELATED PUBLICATIONS

Xu, Yuanping, Xu, Zhijie and Jiang, Xiangqian (2008) 'Category theory-based object-oriented data management for web-based virtual manufacturing', International Journal of Internet Manufacturing and Services, 1 (2), pp. 136-159, ISSN 1751-6048.

Xu, Yuanping, Xu, Zhijie and Jiang, Xiangqian (2008) 'Evaluation on the Categorical DBMS for the VirtualGPS knowledge system', *The Proceedings of computing and engineering annual researchers' conference 2008*, pp.81-86, ISBN 978-1-86218-067-3.

Xu, Yuanping, Xu, Zhijie and Jiang, Xiangqian (2008) 'Implementation of a Category Theory-Based GPS Knowledge System for Product Design', *The 14th International conference on Automation & Computing*, pp.245-249, ISBN978-0-9555293-2-0.

Xu, Yuanping, Xu, Zhijie and Jiang, Xiangqian (2007) 'Machining surface texture knowledge management using a category theory-based object-oriented database', Communications of SIWN (formerly: System and Information Sciences Notes), 1 (1). pp. 83-88. ISSN 1757-4439.

Xu, Yuanping, Xu, Zhijie and Jiang, Xiangqian (2007) 'Exploration of a Category Theory-Based Object-Oriented Database for Surface Texture Information Management', *Proceedings of the First International Symposium on Data, Privacy, and E-Commerce, IEEE Computer Society*, pp. 107-109. ISBN 978-0-7695-3016-1.

REFERENCES

[1] Durakbasa, N. M. and Osanna, P. H. (2001) 'A general approach to workpiece characterization in the frame of GPS (Geometrical Product Specification and Verification)' *International Journal of Machine Tools and Manufacture*, 41, (13-14) pp. 2147-2151

[2] Bennich, P. & Nielsen, H. (2005) *An Overview of GPS – A Cost Saving Tool*. Denmark: Institute for Geometrical Product Specifications

[3] Humienny, Z., et al. (2001) *Geometrical Product Specifications course for Technical Universities*. Warsaw: University of Warsw

[4] International Organisation for Standardisation (1995) ISO/TR 14638:1995 *Geometrical Product Specifications (GPS) –Masterplan*. Geneva: ISO.

[5] International Organisation for Standardisation (2001) ISO TC/213:2001 *Vision Statement*. Geneva: ISO.

[6] Wang, Y., Scott, P. J., and Jiang, X. Q. (2004) The structure of surface texture knowledge, *Journal of Physics: conference series*. 13, pp.1-4 [online] Available from: IOP <http://www.iop.org/EJ/abstract/1742-6596/13/1/001> [Accessed 31 March 2005]

[7] Partridge, D. & Hussain K.M. (1995) *Knowledge Based Information Systems*. London: McGraw-Hill Book Company

[8] Jiang, X. (2004) *A Knowledge-Based Intelligent System for Engineering and Bio-medical Engineering Surface Texture (VirtualSurf)*. Huddersfield: Research project report for University of Huddersfield.

[9] International Organisation for Standardisation (2002) ISO TC/213:2002 *Bussiness plan*. Geneva: ISO.

[10] UGS (2003) *VisVSA guidance* [online] Available at: <http://ugs.com/prdoucts/efactory/docs/fs_visva.pdf> [Accessed 14th May 2006]

[11] Sigmetrix (2003) *What Is CETOL 6 Sigma* [online] Available at: < http://www.sigmetrix.com/about_cetol.asp> [Accessed 15th May 2006]

[12] Carl Zeiss Industrial (2004) *Calypso is CMM software of choice for major automotive manufacturers software* [online] Available at: < http://www.metrologyworld.com/content/news/article.asp?DocID=%7B3BDC736D-B43E-4500-88B2-F4F9099C75FB%7D&Bucket=Supplier+News&VNETCOOKIE=NO> [Accessed 16th May 2006]

[13] Carl Zeiss Industrial (2006) *CMM software* [online] Available at: < http://www.americanmachinist.com/304/Issue/Article/False/8683/Issue> [Accessed 16th May 2006]

[14] Brown & Sharpe, Inc. (2006) *CMM software* [online] Available at: < http://www.brownandsharpe.com/software_overview.asp> [Accessed 16th May 2006]

[15] Martin, J (1977) *Computer Database Organization*. New York: Prentice Hall

[16] Connolly, T.M. and Begg, C.E. (2001) *Database systems: a practical approach to design, implementation, and management*. 3rd ed. London: Addison Wesley

[17] Lin, C. (2003) *Object-Oriented Database Systems: A Survey* [online] Available at: < http://www.cs.man.ac.uk/~david/categories/index.html> [Accessed 9th September 2006]

[18] Molina, G., et al. (2008) *Database Systems: The Complete Book*. 2nd ed. New York: Prentice Hall

[19] Tupil, K. (2008) *Object oriented data model* [online] Available at: <http://www.cs.rpi.edu/tupilk/ooDb/node3.html> [Accessed 10th May 2006]

[20] Cattell, R.G.G., et al. (2000) *The Object Data Standard: ODMG3.0*. San Francisco: Morgan Kaufman

[21] Gray, P.M.D. et al. (1992) *Object-Oriented Databases: A semantic data model Approach*. New Jersey: Prentice Hall International Series in Computer Science

[22] Devarakonda, R.S. (2001) 'Object-Relational Database Systems - The Road Ahead' *Crossroads*, 7, (3) pp.15-18

[23] Stanezyk, S. et al. (1993) *Theory and Practice of Relational Databases*. London: UCL Press Ltd.

[24] Wikipedia (2006) *Comparison of relational database management systems* [online] Available at: < http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems> [Accessed 22nd September 2006]

[25] International Organisation for Standardisation (1996) ISO 12085:1996 *Geometrical Product Specifications (GPS) — Surface texture: Profile method — Motif parameters*. Geneva: ISO.

[26] Fu, T.S. (2002) *Hierarchical modelling of large-scale systems using relational databases*. PhD. thesis, University of Arizona.

[27] Hirao, T. (1990) 'Extension of the relational database semantic processing model' *IBM System Journal*, 29, (4) pp.539-550

[28] Buneman, P. (1997) Functional Database Languages and the Functional Data Model. *In*: A position paper for the FDM workshop, University of Pennsylvania, pp.1-5.

[29] Gray, P.M.G. (1997) *Why we need a Functional Data Model!* [online] Available at: < http://www.csd.abdn.ac.uk/~pgray/fdmwkshop/modeljust.html> [Accessed 14th October 2006]

[30] Embury, S. (1995) *User's Manual for P/FDM V9 Object Database Group*. University of Aberdeen: Dept. of Computing Science.

[31] Intelligent Systems Laboratory (2006) *SICStus Prolog User's Manual*, Sweden: Institute of Computer Science

[32] Nelson, D.A. (1998) *To Formalise and Implement a Categorical Object-Relational Database System*. PhD. Thesis, University of Newcastle upon Tyne.

[33] Levene, M. and Poulovassilis, A. (1991) 'An object-oriented data model formalised through hypergraphs' *Data and Knowledge Engineering*, 6, pp.205-224

[34] Poulovassilis, A. and Levene, M. (1994) 'A nested graph model for the representation and manifpulation of complex objects' *ACM Transactions on Information System*, 12, pp.35-68

[35] Atkinson, M., et al. (1990) The object-oriented database system manifesto. *In*: proceedings of the first international conference on deductive and Object-Oriented Databases, Kyoto, Japan, pp. 223-240.

[36] Committee for Advanced DBMS Function (1990) 'Third generation database system manifesto' *SIGMOD Record*, 19, (3) pp. 31-44

[37] OMG [Object Management Group] (1997) *Object Management Group (OMG)* [online] Available at: < http://www.objs.com/survey/omg.htm> [Accessed 10th January 2007]

[38] Cattell, R.G.G., et al. (1993) *The Object Data Standard:ODMG93, ODMG*. San Francisco: Morgan Kaufman

[39] Cattell, R.G.G., et al. (1997) *The Object Data Standard:ODMG2.0, ODMG*. San Francisco: Morgan Kaufman

[40] OMG [Object Management Group] (1992) *OMG Core Object Model* [online] Available at: < http://www.objs.com/x3h7/omgcore.htm> [Accessed 10th January 2007]

[41] Sun (2008) *Java Data Objects (JDO*) [online] Available at: < http://java.sun.com/jdo/> [Accessed 5th March 2008]

[42] Bancilhon, F., et al. (1992) B*uilding an Object-Oriented Database System: The Story of O2*. London: Morgan Kaufmann

[43] Obasanjo, D. (2001) *An exploration of object oriented database management systems* [online] Available at: < http://www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html> [Accessed 22nd September 2006]

[44] Bagui, S. (2003) 'Achievements and Weaknesses of Object-Oriented Databases' *Journal of Object Technology*, 2, (4) pp.29-41

[45] McClure, S. (1997) *Object Database vs. Object-Relational Databases* [online] Available at: < http://www.geog.ubc.ca/courses/geog470/notes/Object%20Database%20vs_%20Object-Relational%20Databases.htm> [Accessed 25nd September 2006]

[46] Dym, C.L. and Levitt, R.E. (1991) *Knowledge-based systems in engineering.* NewYork: McGraw-Hill

[47] Harmon, P. and King, D. (1985) *Expert Systems – Artificial Intelligence in Business*. New York:Wiley-Interscience

[48] Ackoff, R.L. (1989) 'From Data to Wisdom' *Journal of Applies System Analysis*, 16, pp.3-9

[49] Hopgood, A.A. (2001) *Intelligent System for Engineers and Scientists*. Florida: CRC Press LLC

[50] Tansley, D.S. and Hayball, C.C (1993) *Knowledge-based Systems Analysis and Design − A KADS Developer's Handbook*. London: Prentice Hall

[51] Deitel, H.M., et al. (2003) *Advanced Java 2 Platform How to Program*. 4th ed. London: Pearson Education

[52] Reynolds, M. (2000) *Beginning E-commerce with Visual Basic, ASP, SQL server 7.0 and MTS*. Birmingham: Wrox Press Ltd.

[53] Goguen, J.A. (1989) 'A Categorical Manifesto' *Mathematical Structures in Computer Science*, 1, (1) pp.49-67

[54] Adamek, J., et al. (1990) *Abstract and Concrete Categories: The Joy of Cats.* New York: Wiley-Interscience Pure and Applied Mathematics

[55] Meseguer, J. and Montanari, U. (1988) Petri nets are monoids: A new algebraic foundation for net theory. *In*: proceedings of Symposium on Logic in Computer Science, IEEE Computer Society, pp.155 – 164

[56] Barr, M. and Wells, C. (1996) *Category Theory for Computing Science*. London: Prentice-Hall International Series in Computer Science

[57] Pierce, B. C. (1991) *Basic Category Theory for Computer Scientists*. London: MIT Press

[58] Saunders, M.L. (1998) *Categories for the Working Mathematician*. New York: Springer-Verlg

[59] Rydeheard, D. E. and Burstall, R. M. (2003) *Computational Category Theory* [online] Available at: < http://www.cs.man.ac.uk/~david/categories/index.html> [Accessed 17nd August 2006]

[60] Guo J. (2002) Using Category Theory to Model Software Component Dependencies. *In*: proceedings of the Ninth annual IEEE international conference and workshop on the engineering of computer-based systems, pp.185-192.

[61] Eilenberg, S. and S MacLane, S. (1945) ‘General Theory of Natural Equivalences’ *Transactions of the American Mathematical Society*, 58, (2) pp.231-294

[62] Cartmell, J. (1985) ‘Formalising the Network and Hierarchical Data Models: An Application of Categorical Logic’ *Lecture Notes in Computer Science*, 240, pp.466-492

[63] Ehrich, H.D., et al. (1987) ‘Object, Object Types and Object Identification in categorical methods in Computer Science’ *Lecture Notes in Computer Science*, 393, pp.142-156

[64] Kadish, B. and Diskin, Z. (1997) ‘Algebraic graph-oriented = Category Theory based. Manifesto of categorizing database theory’ *Frame Inform Systems*, Technical Report Series, No.9406, pp.1-10.

[65] Nelson, D.A. and Rossiter, B.N. (1994) *The Categorical Product Data Model as a Formalism for Object-Relational Database*. Newcastle: University of Newcastle upon Tyne (Technical Report Series No. 505).

[66] Nelson, D.A., et al. (1994) *The Functorial Data Model – An Extension to Functional Database*. Newcastle: University of Newcastle upon Tyne (Technical Report Series No. 488).

[67] Hofstede, A.H.M., et al. (1996) ‘Conceptual Data Modelling from a Categorical Perspective’ *The Computer Journal*, 39, (3) pp. 215-231

[68] Colomb, R.M., et al. (2001) ‘Category-Theoretic Fibration as an Abstraction Mechanism in Information Systems’ *Spinger-Verlag*, 38, (1) pp.1-44

[69] Lu, R.Q. (2005) ‘Towards a Mathematical Theory of Knowledge’ *Journal of Computer Science & Technology*, 20, (6) pp.751-757

[70] Kappel, G. and Vieweg, S. (1994) ‘Database Requirements for CIM Applications’ *Information Management in Computer Integrated Manufacturing*, 973, (1995) pp.136-164.

[71] Jacobson, I. (2004) *The unified software development process*. London: Pearson Education

[72] Scott, P.J. (2004), ‘Pattern analysis and metrology: the extraction of stable features from observable measurements’ *Proc.R.Soc.Lond.A*, 460, pp.2845-2864

[73] Scott, P.J. (2006) ‘The case of surface texture parameter RSm’ *Measurement Science and Technology*, 17, pp.559-564

[74] Sowa, J.F. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. London: Thomson Learning

[75] Neggess, J. and Kim, H.S. (1998), *Basic Posets*. London: World Scientific

[76] Rising, L. (1998) *The patterns handbook: techniques, strategies, and applications*. London: Cambridge University Press

[77] Nelson, D. A. and Rossiter, B. N. (1995) Prototyping a Categorical Database in P/FDM. *In*: proceedings of the second international workshop on ADBIS, Springer, pp.247-258.

[78] Objectivity, Inc. (2006) *A comprehensive technical overview of Objectivity/DB and related options* [online] Available at: <http://www.objectivity.com/pages/downloads/whitepaper/pdf/oodb_techOverview.pdf> [Accessed 7th March 2007]

[79] Harold, E.R. (2004) Effective XML. London: Addison-Wesley

[80] Harold, E.R. (2002) P*rocessing XML with Java(TM): A Guide to SAX, DOM, JDOM, JAXP, and TrAX*. London: Addison-Wesley Professional

[81] Db4objects, Inc. (2007) 'db4o Open Source Object Database-product information' *Db4objects, Inc.*, pp.1-8

[82] ETH (2007), *Object-Oriented Databases db4o: Part 1* [online] Available at: <http://www.odbms.org/download/035.03%20Grossniklaus%20ODBMS%20Lecture%20-%20db4o%20Part1%20November%202007.PDF>[Accessed 8th May 2008]

[83] Db4objects, Inc. (2008) *Db4o description* [online] Available at: < http://handheld.softpedia.com/get/Developer-Tools/db4o-40582.shtml> [Accessed 24th February 2008]

[84] Paterson, J., et al., (2006) *The Definitive Guide to db4o*, California: Apress Berkely

[85] Db4objects, Inc. (2005) *The Business Case for the Open Source Object Database* [online] Available at: < http://www.db4o.com/about/company/backgrounder/db4objects%20Business%20Backgrounder%20June%202005.pdf> [Accessed 25th December 2007]

[86] DB4O Developer Community (2008) *db4o news and press releases*, [online] Available at: < http://developer.db4o.com/blogs/db4o_news_and_press_releases/default.aspx> [Accessed 25th July 2008]

[87] Java Net (2007) JavaCC [online] Available at: <https://javacc.dev.java.net/> [Accessed 17th October 2007]

[88] Tsichritzis, D. and Klug, A. (1978) 'The ANSI/X3/SPARC DBMS Framework' *Information System*, 3, (3) pp.173-191

[89] SDN [Sun Developer Network] (2005) *Java Programming-accessors, mutators, SDN* [online] Available at: < http://forums.sun.com/thread.jspa?threadID=683522&messageID=3981321> [Accessed 17th October 2007]

[90] Ambler, S.W. (2000) *The whys and why nots Of Java accessors* [online] Available at: < http://www.ibm.com/developerworks/java/library/ws-tip-why.html> [Accessed 18th October 2007]

[91] Mummery, L. (1990) *Surface Texture Analysis - The Handbook*. Germany: Hommelwerke.

[92] Kalpakjian, S. and Schmid, S. (2005) *Manufacturing Engineering and Technology*. London: Prentice Hall

[93] Whitehouse, D. J. (2002), *Surfaces and their measurement*. London: Hermes Penton Ltd.

[94] Filetin, T. (2002) *The knowledge bases for selecting the surface modification treatment*. MT2002

[95] International Organisation for Standardisation (2002) ISO 1302:2002 *Geometrical Product Specifications (GPS) — Indication of surface texture in technical product documentation*. Geneva: ISO.

[96] International Organisation for Standardisation (1997) ISO 4287:1997 *Geometrical Product Specifications (GPS) — Surface texture: Profile method — Terms, definitions and surface texture parameters*. Geneva: ISO.

[97] International Organisation for Standardisation (1996) ISO 13565-2:1996 *Geometrical Product Specifications (GPS) — Surface texture: Profile method; Surfaces having stratified functional properties — Part 2: Height characterization using the linear material ratio curve*. Geneva: ISO.

[98] International Organisation for Standardisation (2000) ISO 13565-3:2000 *Geometrical Product Specifications (GPS) — Surface texture: Profile method; Surfaces having stratified functional properties — Part 3: Height characterization using the material probability curve*. Geneva: ISO.

[99] International Organization for Standardization (1996) ISO 4288:1996 *Geometrical Product Specifications (GPS) -Surface texture: Profile method - Rules and procedures for the assessment of surface texture*. ISO: Geneva.

[100] International Organisation for Standardisation (1996) ISO 13565-1:1996 Geometrical *Product Specifications (GPS) — Surface texture: Profile method; Surfaces having stratified functional properties — Part 1: Filtering and general measurement conditions*. Geneva:ISO.

[101] International Organisation for Standardisation (1996) ISO 3274:1996 *Geometrical Product Specifications (GPS) - Surface texture: Profile method - Nominal characteristics of contact (stylus) instruments*. Geneva:ISO.

[102] Schey, J (2000) *Introduction to manufacturing processes*. London: McGraw Hill

[103] Swift, K.G. & Booker, J.D. (2003) *Process selection: from design to manufacture*. London Elsevier Butterworth-Heinemann

[104] Griffiths, B. (2001) *Manufacturing Surface Technology – Surface Integrity & Functional Performance*. London: Penton Press

[105] British Standards Institution (1990) BS 1134-2:1990 *Assessment of surface texture — Part 2: Guidance and general information*. London:BSI.

[106] Leach, R. (2001) *Measurement Good Practise Guide No.37 – The Measurement of Surface Texture using Stylus Instruments*. London: National Physical Laboratory.

[107] Whitehouse, D. J. (1997) 'Surface metrology' *Measurement Science and Technology*, 8, pp.955-972

[108] Sun (2007) *Java 2 SDK 1.4.2 Installation Notes for Microsoft Windows* [online] Available at: <http://java.sun.com/j2se/1.4.2/install-windows.html> [Accessed 18th October 2007]

[109] Erickson M. and McIntyre, A. (2001) *What is Eclipse, and how do I use it?* [online] Available at: < http://www.ibm.com/developerworks/opensource/library/os-eclipse.html> [Accessed 20th October 2007]

[110] Eclipse (2007) *Eclipse Plugin Central (EPIC)* [online] Available at: < http://www.eclipseplugincentral.com/> [Accessed 20th October 2007]

[111] JFreeChart (2007) *Welcome To JFreeChart!* [online] Available at: < http://www.jfree.org/jfreechart/> [Accessed 10th November 2007]

[112] Eclipse (2005) *Eclipse for Java and Web Developers* [online] Available at: < http://www.eclipse.org/callisto/java.php/> [Accessed 20th October 2007]

[113] Gordon, A.D. and Hammond, K. (1995) Monadic I/O in Haskell 1.3. *In*: proceedings of the haskell workshop, California, pp.50-68

APPENDIX A – CODE LIST FOR “*MEASURAND*” CLASS CATEGORY

```
import cpt.ctdb.dataModel.*;
public class Measurand extends Category{

    public Arrow interObjId_id;
      public Arrow interObjId_tolerance_type;
     public Arrow interObjId_parameter_type;
     public Arrow interObjId_parameter_name;
    public Arrow interObjId_machine_allowance;
  public Arrow interObjId_parameterExtends;

   public void setArrows(Arrow interObjId_id, Arrow
         interObjId_tolerance_type, Arrow interObjId_parameter_type, Arrow
         interObjId_parameter_name, Arrow interObjId_machine_allowance,
         Arrow interObjId_parameterExtends){
           this.interObjId_id= interObjId_id;
           this.interObjId_tolerance_type = interObjId_tolerance_type;
           this.interObjId_parameter_type = interObjId_parameter_type;
           this.interObjId_parameter_name = interObjId_parameter_name;
           this.interObjId_machine_allowance =
                                             interObjId_machine_allowance;
          this.interObjId_parameterExtends = interObjId_parameterExtends;
   }

   public void setTargetForIdArrow(int id){
          this.interObjId_id.setTarget(Integer.valueOf(id));
   }

   public void setTargetForTolerTypeArrow(String tolerance_type){
          //inference rules for setting the default tolerance type.
    }

   public void setTargetForParaTypAndparaNameArrow(String
                                                     parameter_type){
       //inference rules for setting the default parameter type and parameter
       //name.
   }

    public String setTargetForparameterExtendsArrow(double value){
           //inference rules for setting the parameter extends according to
         //user inputted parameter value.
   }

    public Arrow getIdArrow(){
          return this.interObjId_id;
    }

    public Arrow getTolerTypeArrow(){
           return this.interObjId_tolerance_type;
    }

    public Arrow getParaTypeArrow(){
           return this.interObjId_parameter_type;
    }

    public Arrow getParaNameArrow(){
           return this.interObjId_parameter_name;
```

```
        }

        public Arrow getParaExtendArrow(){
                return this.interObjId_parameterExtends;
        }

        public Arrow getMachineAllowArrow(){
                return this.interObjId_machine_allowance;
        }

       public String toString(){
            return ("Measurand
            ["+((Integer)getIdArrow().getTarget()).intValue()+"]:
            tolerance_type="+((String)getTolerTypeArrow().getTarget())+"
            parameter_type="+((String)getParaTypeArrow().getTarget())+"
            parameter_name="+((String)getParaNameArrow().getTarget())+"
            parameter_value_extend="+((String)getParaExtendArrow().getTarget
            ())+"
            machine_allowance="+((Double)getMachineAllowArrow().getTarget(
            ))).toString();
       }
}
```

APPENDIX B – CODE LIST FOR "*CALLOUT*" CLASS CATEGORY

```
import cpt.ctdb.dataModel.Arrow;
import cpt.ctdb.dataModel.RelationConstraint;
public class Callout extends PersistCategory {

     public Arrow interObjId_id;
     public Arrow interObjId_partition_dirSym;
     public Arrow interObjId_partition_manuTypSym;
     public Arrow interObjId_partition_manuMethod;
    public Arrow interObjId_extraction_numCutOff;
   public Arrow interObjId_extraction_sampLength;
   public Arrow interObjId_extraction_evaLength;
   public Arrow interObjId_filtration_filterType;
   public Arrow interObjId_filtration_upLimit;
   public Arrow interObjId_filtration_lowLimit;
   public Arrow interObjId_measurand_tolerType;
   public Arrow interObjId_measurand_paraType;
   public Arrow interObjId_measurand_machineAllow;
  public Arrow interObjId_measurand_paraExtends;
  public Arrow interObjId_limitedValue;


  public void setArrows(Arrow interObjId_id, Arrow
      interObjId_partition_dirSym, Arrow interObjId_partition_manuTypSym,
      Arrow interObjId_partition_manuMethod, Arrow
      interObjId_extraction_numCutOff, Arrow
      interObjId_extraction_sampLength, Arrow
      interObjId_extraction_evaLength, Arrow
      interObjId_filtration_FilterType, Arrow interObjId_filtration_upLimit,
      Arrow interObjId_filtration_lowLimit, Arrow
      interObjId_measurand_tolerType, Arrow
      interObjId_measurand_paraType, Arrow
      interObjId_measurand_machineAllow, Arrow interObjId_limitedValue,
      Arrow interObjId_measurand_paraExtends){
      this.interObjId_id= interObjId_id;
      this.interObjId_partition_dirSym = interObjId_partition_dirSym;
     this.interObjId_partition_manuTypSym =
                                        interObjId_partition_manuTypSym;
    this.interObjId_partition_manuMethod =
                                  interObjId_partition_manuMethod;
    this.interObjId_extraction_numCutOff =
                                  interObjId_extraction_numCutOff;
    this.interObjId_extraction_sampLength =
                                  interObjId_extraction_sampLength;
   this.interObjId_extraction_evaLength = interObjId_extraction_evaLength;
   this.interObjId_filtration_filterType = interObjId_filtration_FilterType;

  this.interObjId_filtration_upLimit = interObjId_filtration_upLimit;
   this.interObjId_filtration_lowLimit =  interObjId_filtration_lowLimit;
  this.interObjId_measurand_tolerType = interObjId_measurand_tolerType;
  this.interObjId_measurand_paraType = interObjId_measurand_paraType;
  this.interObjId_measurand_machineAllow =
                                interObjId_measurand_machineAllow;
  this.interObjId_limitedValue = interObjId_limitedValue;
  this.interObjId_measurand_paraExtends =
                                interObjId_measurand_paraExtends;
  }
  public void setTargetForIdArrow(int id){
         this.interObjId_id.setTarget(Integer.valueOf(id));
  }

  public void setTargetForPartitionDirSymArrow(String direction_symbol){
         this.interObjId_partition_dirSym.setTarget(direction_symbol);
  }

  public void setTargetForPartitionmanuTypSymArrow(String
                                               manufacture_type_symbol){
        this.interObjId_partition_manuTypSym.setTarget(manufacture_type_sy
                                                        mbol);
   }

public void setTargetForPartitionmanuMethodArrow(String
                                                  manufacture_method){
     this.interObjId_partition_manuMethod.setTarget(manufacture_method);
  }

  public void setTargetForExtractionNumCutOffArrow(Integer num_cutOff){
      this.interObjId_extraction_numCutOff.setTarget(num_cutOff);
  }

  public void setTargetForExtractionSampLengthArrow(Double
                                                      sampling_length){
        this.interObjId_extraction_sampLength.setTarget(sampling_length);
  }

  public void setTargetForExtractionEvaLengthArrow(Double
                                                  evaluation_Length){
         this.interObjId_extraction_evaLength.setTarget(evaluation_Length);
 }

  public void setTargetForFiltrationFilterTypeArrow(String filter_type){
        this.interObjId_filtration_upLimit.setTarget(filter_type);
  }

  public void setTargetForFiltrationUpLimitArrow(Double up_limit){
         this.interObjId_filtration_upLimit.setTarget(up_limit);
   }

  public void setTargetForFiltrationLowLimitArrow(Double low_limit){
         this.interObjId_filtration_lowLimit.setTarget(low_limit);
  }

  public void setTargetForMeasurandTolerTypeArrow(String
                                                  tolerance_type){
        this.interObjId_measurand_tolerType .setTarget(tolerance_type);
 }

  public void setTargetForMeasurandParaTypeArrow(String
                                                 parameter_type){
     this.interObjId_measurand_paraType.setTarget(parameter_type);
  }

  public void setTargetForMeasurandMachineAllowArrow(Double
                                                   machine_allowance){
        this.interObjId_measurand_machineAllow.setTarget(machine_allowan
                                                        ce);
        }

        public void setTargetForlimitedValueArrow(double limitedValue){
            this.interObjId_limitedValue.setTarget(Double.valueOf(limitedValue));
        }

        public Arrow getIDArrow(){
            return this.interObjId_id;
        }

        public Arrow getPartitionDirSymArrow(){
                return this.interObjId_partition_dirSym;
        }

         public Arrow getPartitionManuMethodArrow(){
                return this.interObjId_partition_manuTypSym;
         }

         public Arrow getPartitionManuTypSymArrow(){
                return this.interObjId_partition_manuMethod;
        }

        public Arrow getExtractionNumCutOffArrow(){
                return this.interObjId_extraction_numCutOff;
         }

        public Arrow getExtractionSampLengthArrow(){
                return this.interObjId_extraction_sampLength;
        }

         public Arrow getExtractionEvaLengthArrow(){
                 return this.interObjId_extraction_evaLength;
         }

        public Arrow getFiltrationFilterTypeArrow(){
                return this.interObjId_filtration_filterType;
        }

        …………

         public Arrow getMeasurandTolerTypeArrow(){
                return this.interObjId_measurand_tolerType;
        }

        public Arrow getMeasurandParaTypeArrow(){
                return this.interObjId_measurand_paraType;
        }

         public Arrow getMeasurandMachineAllowArrow(){
                return this.interObjId_measurand_machineAllow;
         }

         public Arrow getMeasurandLimitValueArrow(){
                return this.interObjId_limitedValue;
         }

        public String toString(){
                        ………………
        }
}
```

APPENDIX C – CODE LIST FOR "*PRODUCTFORCALLOUT*" CLASS CATEGORY

```
package surfaceTexture;

    import cpt.ctdb.dataModel.*;
    public class ProductForCallout extends Product{
           public Arrow interObjId_id;
           Functor calloutToMeasurand;
           Functor calloutToExtraction;
           Functor calloutToFiltration;
           Functor calloutToPartition;

           public void setFunctors(Functor calloutToMeasurand, Functor
              calloutToExtraction, Functor calloutToFiltration, Functor
              calloutToPartition){
               this.calloutToMeasurand= calloutToMeasurand;
               this.calloutToExtraction = calloutToExtraction;
               this.calloutToFiltration = calloutToFiltration;
               this.calloutToPartition = calloutToPartition;
         }

         public void setTargetForIdArrow(int id){
                this.interObjId_id.setTarget(Integer.valueOf(id));
         }

         public Arrow getIdArrow(){
                return this.interObjId_id;
         }

         public Functor getCalloutToMeasurand(){
               return this.calloutToMeasurand;
         }

        public Functor getCalloutToExtraction(){
               return this.calloutToExtraction;
        }

       public Functor getCalloutToFiltration(){
               return this.calloutToFiltration;
       }

       public Functor getCalloutToPartition(){
               return this.calloutToPartition;
       }

       public boolean checkMonomorphismForMeasurand(){
             CTDBObjectSet result = getAllInstances(Measurand.class);
              CTDBObjectSet result1 =
                                    getAllInstances(ProductForCallout.class
                                    );
             for (int a=0; a<= result1.size(); a++){
                   int j=0;
                for (int b=0; b<= result1.size(); b++){
                   if(((Functor)((ProductForCallout)result1.next()).getCalloutT
                   oMeasurand()).getTarget().getObjectInternalId()==
                   ((Measurand)result.next()).getObjectInternalId()){
                          j++;
                    }
                }
                if (j>1){
                    return false;
                }
            }
            return true;
        }

        public boolean checkEpimorphismForMeasurand(){
                CTDBObjectSet result = getAllInstances(Measurand.class);
                 CTDBObjectSet result1 =
                                    getAllInstances(ProductForCallout.class);
                for (int a=0; a<= result1.size(); a++){
                        int j=0;
                     for (int b=0; b<= result1.size(); b++){
                         if(((Functor)((ProductForCallout)result1.next()).getCallout
                         ToMeasurand()).getTarget().getObjectInternalId()==
                         ((Measurand)result.next()).getObjectInternalId()){
                                 j++;
                           }
                      }
                     if (j==0){
                            return false;
                     }
                 }
                return true;
         }

        public boolean checkIsomorphismForMeasurand(){
            if(checkMonomorphismForMeasurand()&&checkEpimorphismForMeasurand()){
                 return true;
              }
              return false;
        }
}
```

APPENDIX D – CODE LIST FOR CREATING A DATABASE SCHEMA FOR CALLOUT

```
Measurand m = new Measurand();
        ……………
//creating and populating instances for "Measurand".
Extraction e=new Extraction();
……………
//creating and populating instance for "Extraction".
Filtration f=new Filtration();
……………
//creating and populating instance for "Filtration".
Partition p= new Partition();
……………
//creating and populating instance for "Partition".
Callout callout= new Callout();
……………
//creating and populating instance for "Callout".
ProductForCallout productForcallout = new ProductForCallout();
productForcallout. setTargetForIdArrow(1);
productForcallout.setName("pullback_callout");
productForcallout.setAry(4);
productForcallout.setVertex(callout);
Functor calloutToMeasurand = new Functor();
……………
// populating instance for functor mapping from instance of "Callout" to
//"Measurand".
Functor calloutToExtraction = new Functor();
……………
// populating instance for functor mapping from instance of "Callout" to
//"Extraction".
Functor calloutToFiltration = new Functor();
……………
// populating instance for functor mapping from instance of "Callout" to
//"Filtration".
Functor calloutToPartition = new Functor();
……………
// populating instance for functor mapping from instance of "Callout" to
//"Partition".
productForcallout.setFunctors(calloutToMeasurand,                    calloutToExtraction,
                              calloutToFiltration, calloutToPartition)
db.set(productForcallout);
```

APPENDIX E − MANUFACTURING PROCESS PRIMA SELECTION MATRIX

Quantity / Material	Very low 1 to 100	Low 100 to 1,000	Low to medium 1,000 to 10,000	Medium to high 10,000 to 100,000	High 100,000+	All quantities
Irons	[1.5] [1.6] [1.7] [4.M]	[1.2] [1.5] [1.6] [1.7] [4.M] [5.3] [5.4]	[1.2] [1.3] [1.5] [1.6] [1.7] [3.11] [4.A] [5.2]	[1.2] [1.3] [3.11] [4.A]	[1.2] [1.3] [3.11] [4.A]	[1.1]
Steel (carbon)	[1.5] [1.7] [3.10] [4.M] [5.1] [5.5] [5.6]	[1.2] [1.5] [1.7] [3.10] [4.M] [5.1] [5.3] [5.4] [5.5]	[1.2] [1.3] [1.5] [1.7] [3.1] [3.3] [3.10] [3.11] [4.A] [5.2] [5.3] [5.4] [5.5]	[1.9] [3.1] [3.3] [3.4] [3.5] [3.11] [3.12] [4.A] [5.2] [5.5]	[1.9] [3.1] [3.2] [3.3] [3.4] [3.5] [3.12] [4.A]	[1.1] [1.6] [3.6] [3.8] [3.9]
Steel (tool, alloy)	[1.1] [1.5] [1.7] [3.10] [4.M] [5.1] [5.5] [5.6] [5.7]	[1.1] [1.2] [1.7] [4.M] [5.1] [5.3] [5.4] [5.5] [5.6] [5.7]	[1.2] [1.5] [1.7] [3.1] [3.4] [3.11] [4.A] [5.2] [5.3] [5.4] [5.5]	[3.1] [3.4] [3.5] [3.11] [3.12] [4.A] [5.2]	[4.A]	[1.6] [3.6]
Stainless steel	[1.5] [1.7] [3.7] [3.10] [4.M] [5.1] [5.5] [5.6]	[1.2] [1.7] [3.7] [3.10] [4.M] [5.1] [5.3] [5.4] [5.5]	[1.2] [1.5] [1.7] [3.1] [3.3] [3.7] [3.10] [3.11] [4.A] [5.2] [5.3] [5.4] [5.5]	[1.9] [3.1] [3.3] [3.4] [3.5] [3.11] [3.12] [4.A]	[1.9] [3.2] [3.3] [4.A]	[1.1] [1.6] [3.6] [3.8] [3.9]
Copper & alloys	[1.5] [1.7] [3.10] [4.M] [5.1]	[1.2] [1.5] [1.7] [1.8] [3.5] [3.10] [4.M] [5.1] [5.3] [5.4]	[1.2] [1.3] [1.5] [1.8] [3.1] [3.3] [3.10] [3.11] [4.A] [5.2] [5.3] [5.4]	[1.2] [1.4] [1.9] [3.1] [3.3] [3.4] [3.5] [3.11] [3.12] [4.A]	[1.2] [1.9] [3.1] [3.2] [3.3] [3.4] [3.5] [3.7] [3.8] [3.11] [3.12] [4.A]	[1.1] [1.6] [3.6] [3.8] [3.9] [5.5]
Aluminium & alloys	[1.5] [1.7] [3.7] [3.10] [4.M] [5.5]	[1.2] [1.5] [1.7] [1.8] [3.7] [3.10] [4.M] [5.3] [5.4] [5.5]	[1.2] [1.3] [1.5] [1.8] [3.1] [3.3] [3.7] [3.10] [3.11] [4.A] [5.3] [5.4] [5.5]	[1.2] [1.3] [1.4] [1.9] [3.1] [3.3] [3.4] [3.5] [3.11] [3.12] [4.A] [5.5]	[1.2] [1.3] [1.4] [1.9] [3.1] [3.2] [3.3] [3.4] [3.5] [3.8] [3.12] [4.A]	[1.1] [1.6] [3.6] [3.8] [3.9]
Magnesium & alloys	[1.6] [1.7] [3.10] [4.M] [5.1] [5.5]	[1.6] [1.7] [1.8] [3.10] [4.M] [5.5]	[1.3] [1.6] [1.8] [3.1] [3.3] [3.4] [3.10] [4.A] [5.5]	[1.3] [1.4] [3.1] [3.3] [3.4] [3.5] [3.12] [4.A]	[1.3] [1.4] [3.1] [3.3] [3.4] [3.8] [3.12] [4.A]	[1.1] [3.6] [3.8] [3.9]
Zinc & alloys	[1.1] [1.7] [3.10] [4.M] [5.5]	[1.1] [1.7] [1.8] [3.10] [4.M] [5.5]	[1.3] [1.8] [3.3] [3.10] [4.A] [5.5]	[1.3] [1.4] [3.3] [3.4] [3.5] [3.12] [4.A]	[1.4] [3.2] [3.3] [3.4] [3.5] [4.A]	[3.6] [3.8] [3.9]
Tin & alloys	[1.1] [1.7] [3.10] [4.M] [5.5]	[1.1] [1.7] [1.8] [3.10] [4.M] [5.5]	[1.3] [1.8] [3.3] [3.10]	[1.3] [1.4] [3.3] [3.4] [3.12]	[1.4] [3.3] [3.4] [4.A]	
Lead & alloys	[1.1] [3.10] [4.M] [5.5]	[1.1] [1.8] [3.10] [4.M] [5.5]	[1.1] [1.8] [3.3] [3.10]	[1.3] [1.4] [3.3] [3.4] [3.5] [3.12] [4.A]	[1.4] [3.2] [3.3] [3.4] [4.A]	[3.6]
Nickel & alloys	[1.5] [1.7] [3.10] [4.M] [5.1] [5.5] [5.6]	[1.2] [1.5] [1.7] [3.10] [4.M] [5.1] [5.3] [5.4] [5.5]	[1.2] [1.3] [1.5] [1.7] [3.1] [3.3] [3.11] [4.A] [5.2] [5.3] [5.4] [5.5]	[3.1] [3.3] [3.4] [3.5] [3.11] [3.12] [4.A] [5.2] [5.5]	[3.2] [3.3] [4.A]	[1.1] [1.6] [3.6] [3.8] [3.9]

			[3.10]			
Titanium & alloys	[1.1] [1.6] [3.7] [3.10] [4.M] [5.1] [5.5] [5.6] [5.7]	[1.1] [1.6] [3.7] [3.10] [4.M] [5.1] [5.3] [5.4] [5.5] [5.6] [5.7]	[3.1] [3.7] [3.10] [3.11] [4.A] [5.2] [5.3] [5.4] [5.5]	[3.1] [3.4] [3.11] [3.12] [4.A] [5.2] [5.5]	[4.A]	[3.8] [3.9]
Thermo plastics	[2.5] [2.7]	[2.3] [2.5] [2.7]	[2.3] [2.5] [2.6] [2.7]	[2.1] [2.3] [2.5] [2.6] [2.9]	[2.1] [2.6] [2.9]	
Thermo sets	[2.5] [2.7]	[2.2] [2.3]	[2.2] [2.3] [2.4]	[2.1] [2.3] [2.9]	[2.1] [2.3] [2.4] [2.9]	
Fr composites	[2.2] [2.8] [5.7]	[2.2] [2.3] [2.8] [5.7]	[2.1] [2.2] [2.3]	[2.1] [2.3]		
Ceramics	[1.5] [5.1] [5.5] [5.6] [5.7]	[5.1] [5.3] [5.5] [5.6] [5.7]	[5.2] [5.3] [5.4] [5.5]	[3.11]	[3.7] [3.11]	[5.5]
Refractory metals	[1.1] [5.7]	[5.7]		[3.12]		[1.6]
Precious metals	[5.5]	[5.5]	[5.5]	[3.5]	[3.5]	[1.6]

Key to manufacturing process PRIMA selection matrix:
[1.1] Sand casting
[1.2] Shell moulding
[1.3] Gravity die casting
[1.4] Pressure die casting
[1.5] Centrifugal casting
[1.6] Investment casting
[1.7] Ceramic mould casting
[1.8] Plaster mould casting
[1.9] Squeeze casting

[2.1] Injection moulding
[2.2] Reaction injection moulding
[2.3] Compression moulding
[2.4] Transfer moulding
[2.5] Vacuum forming
[2.6] Blow moulding
[2.7] Rotational moulding
[2.8] Contact moulding
[2.9] Continuous extrusion (plastics)

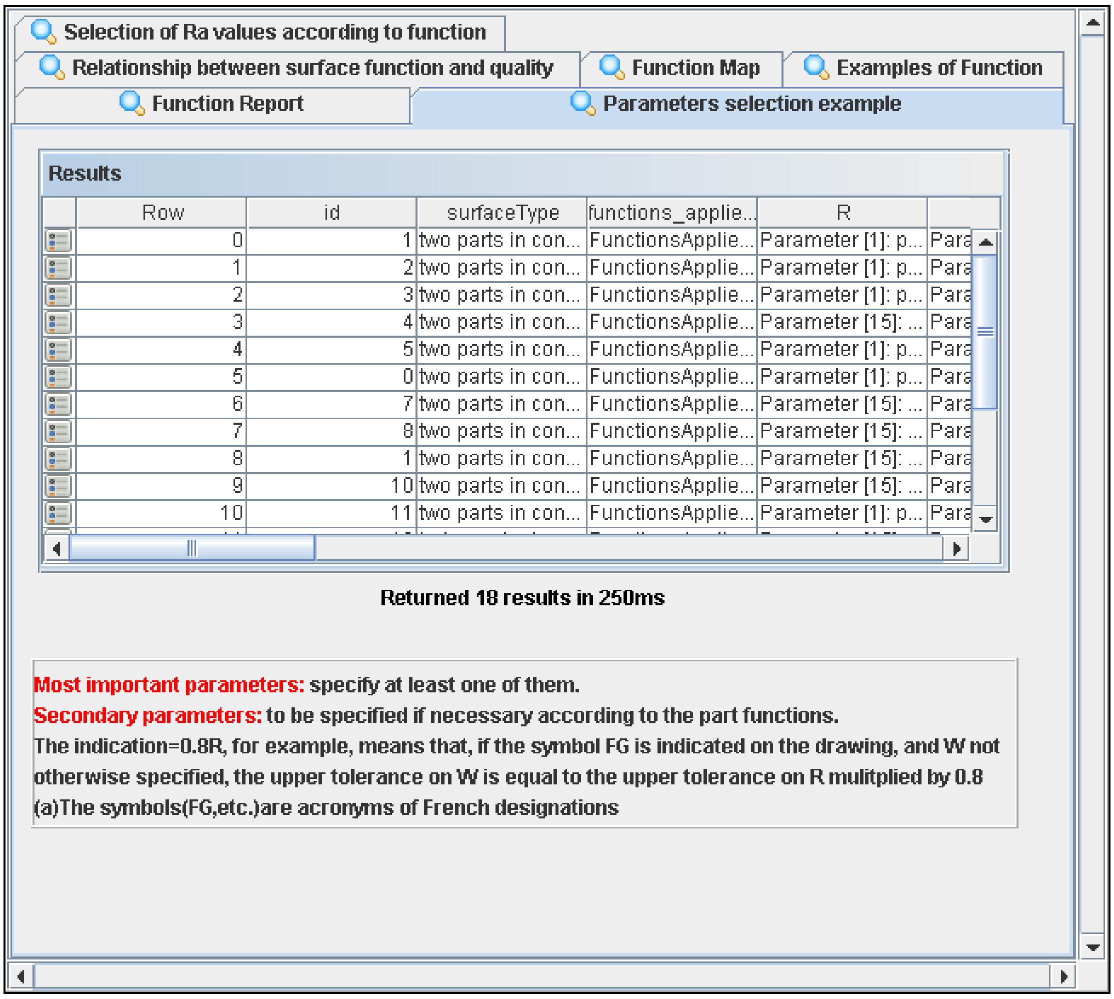
[3.1] Closed die forging
[3.2] Rolling
[3.3] Drawing
[3.4] Cold forming
[3.5] Cold heading
[3.6] Swaging
[3.7] Superplastic forming
[3.8] Sheet-metal shearing
[3.9] Sheet-metal forming
[3.10] Spinning
[3.11] Powder metallurgy
[3.12] Continuous extrusion (metals)

[4.A] Automatic machining
[4.M] Manual machining

[5.1] Electrical discharge machining (EDM)
[5.2] Electrochemical machining (ECM)

[5.3] Electron beam machining (EBM)
[5.4] Laser beam machining (LBM)
[5.5] Chemical Machining (CM)
[5.6] Ultrasonic machining (USM)
[5.7] Abrasive jet machining (AJM)

APPENDIX F – PARAMETERS SELECTION EXAMPLE


Selection of Ra values according to function
Relationship between surface function and quality
Function Map
Examples of Function
Function Report
Parameters selection example

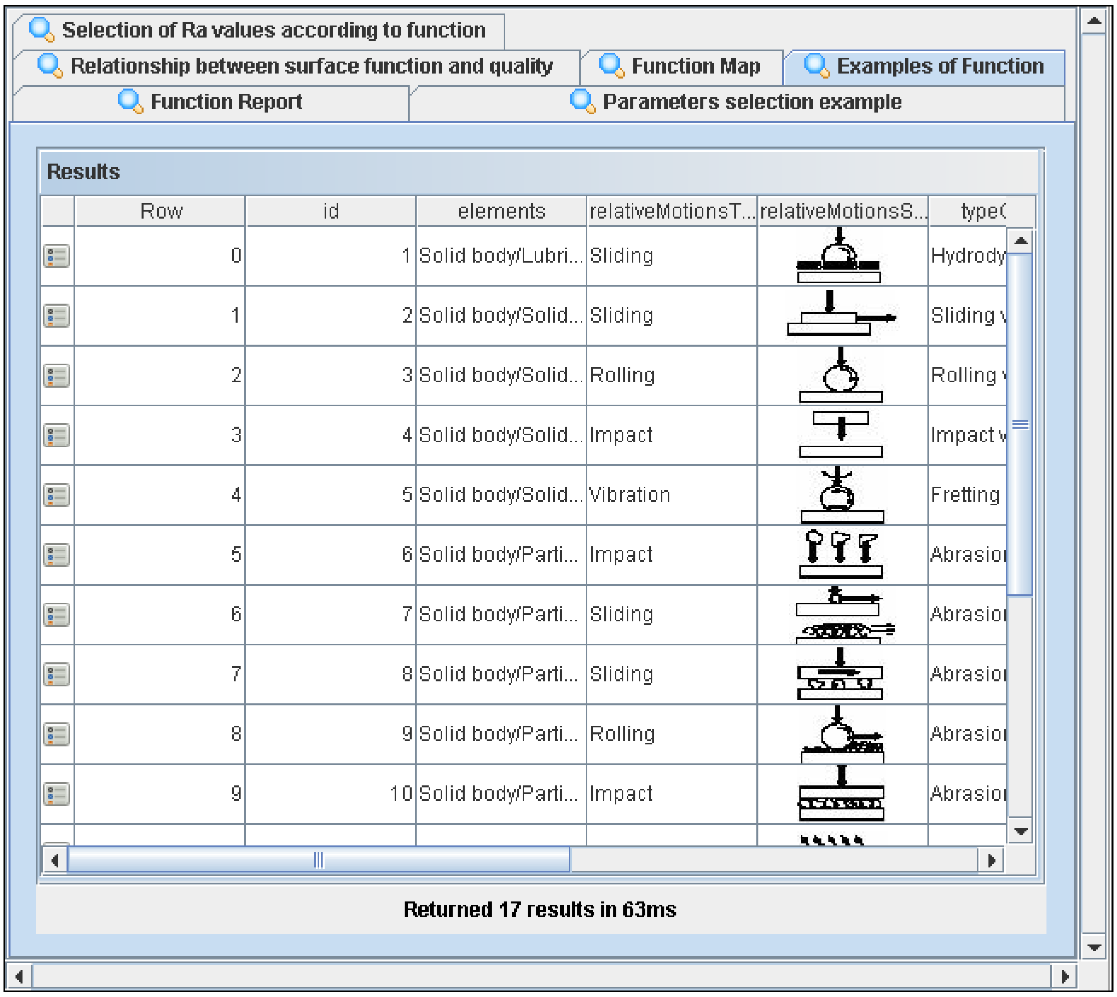
Results

Row	id	surfaceType	functions_applie...	R	
0	1	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para
1	2	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para
2	3	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para
3	4	two parts in con...	FunctionsApplie...	Parameter [15]: ...	Para
4	5	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para
5	0	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para
6	7	two parts in con...	FunctionsApplie...	Parameter [15]: ...	Para
7	8	two parts in con...	FunctionsApplie...	Parameter [15]: ...	Para
8	1	two parts in con...	FunctionsApplie...	Parameter [15]: ...	Para
9	10	two parts in con...	FunctionsApplie...	Parameter [15]: ...	Para
10	11	two parts in con...	FunctionsApplie...	Parameter [1]: p...	Para

Returned 18 results in 250ms

Most important parameters: specify at least one of them.
Secondary parameters: to be specified if necessary according to the part functions.
The indication=0.8R, for example, means that, if the symbol FG is indicated on the drawing, and W not otherwise specified, the upper tolerance on W is equal to the upper tolerance on R mulitplied by 0.8
(a)The symbols(FG,etc.)are acronyms of French designations

APPENDIX G – EXAMPLE OF FUNCTION CORELATION



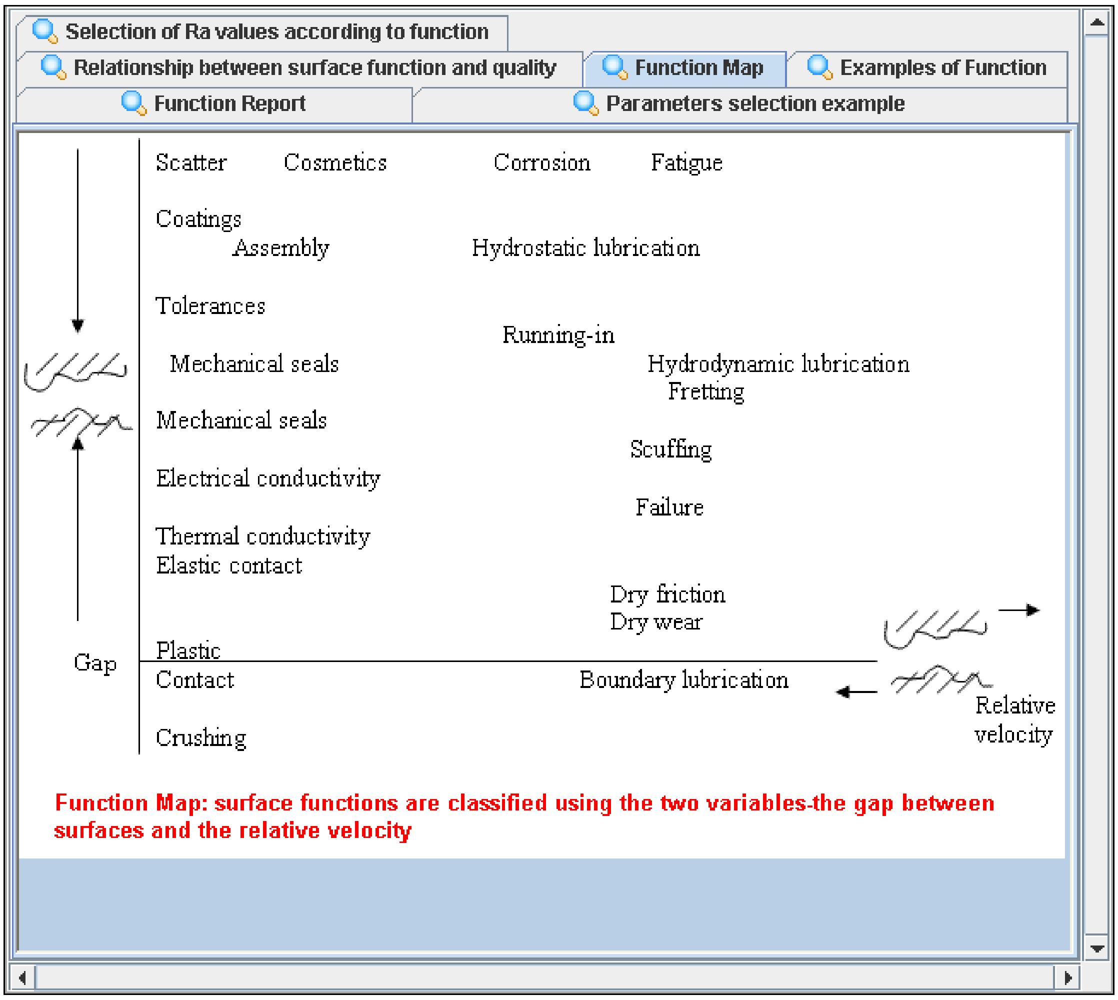
Selection of Ra values according to function | Relationship between surface function and quality | Function Map | Examples of Function | Function Report | Parameters selection example

Results

	Row	id	elements	relativeMotionsT...	relativeMotionsS...	type(
	0	1	Solid body/Lubri...	Sliding		Hydrody
	1	2	Solid body/Solid...	Sliding		Sliding v
	2	3	Solid body/Solid...	Rolling		Rolling v
	3	4	Solid body/Solid...	Impact		Impact w
	4	5	Solid body/Solid...	Vibration		Fretting
	5	6	Solid body/Parti...	Impact		Abrasio
	6	7	Solid body/Parti...	Sliding		Abrasio
	7	8	Solid body/Parti...	Sliding		Abrasio
	8	9	Solid body/Parti...	Rolling		Abrasio
	9	10	Solid body/Parti...	Impact		Abrasio

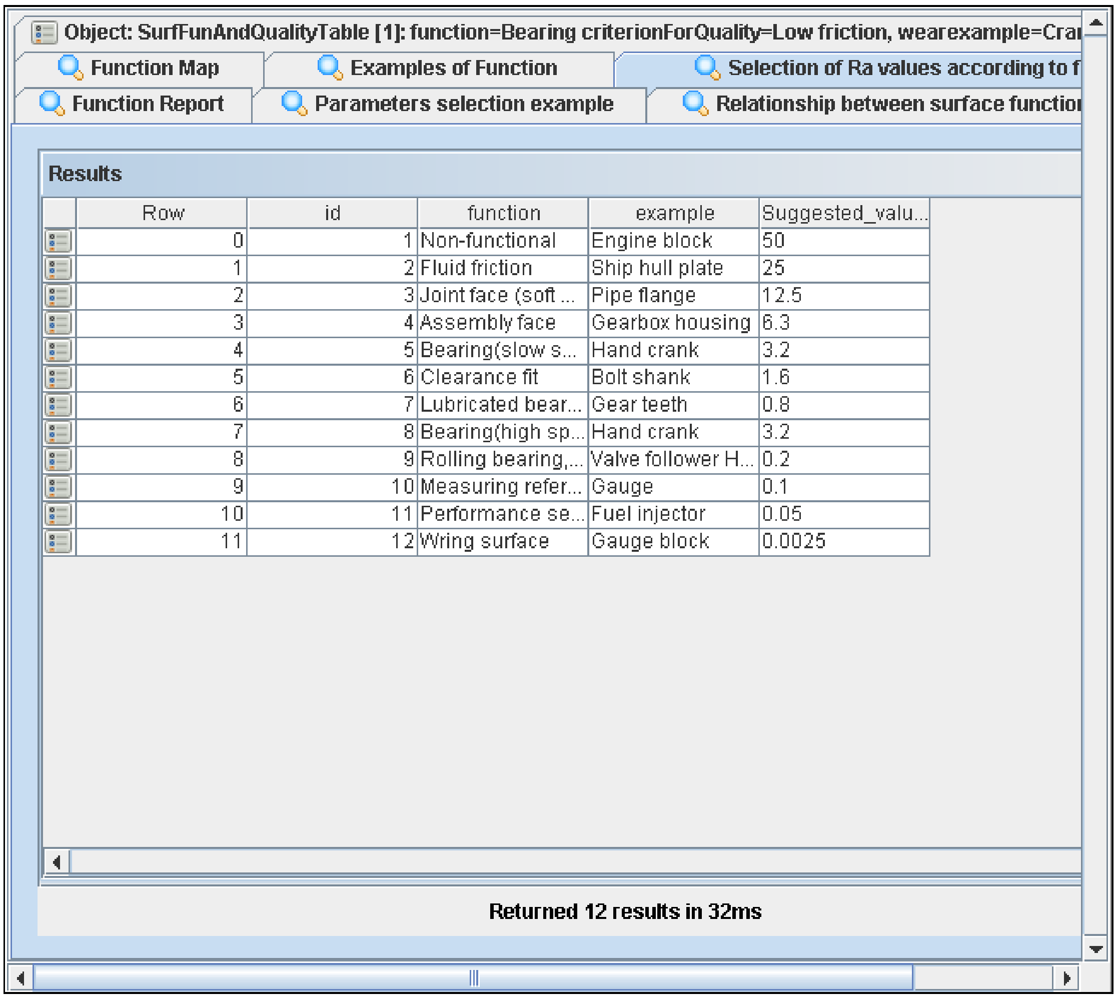
Returned 17 results in 63ms

APPENDIX H – FUNCTION MAP



Function Map: surface functions are classified using the two variables-the gap between surfaces and the relative velocity

APPENDIX I – SELECTION OF *Ra* ACCORDING TO FUNCTION



Object: SurfFunAndQualityTable [1]: function=Bearing criterionForQuality=Low friction, wearexample=Cra

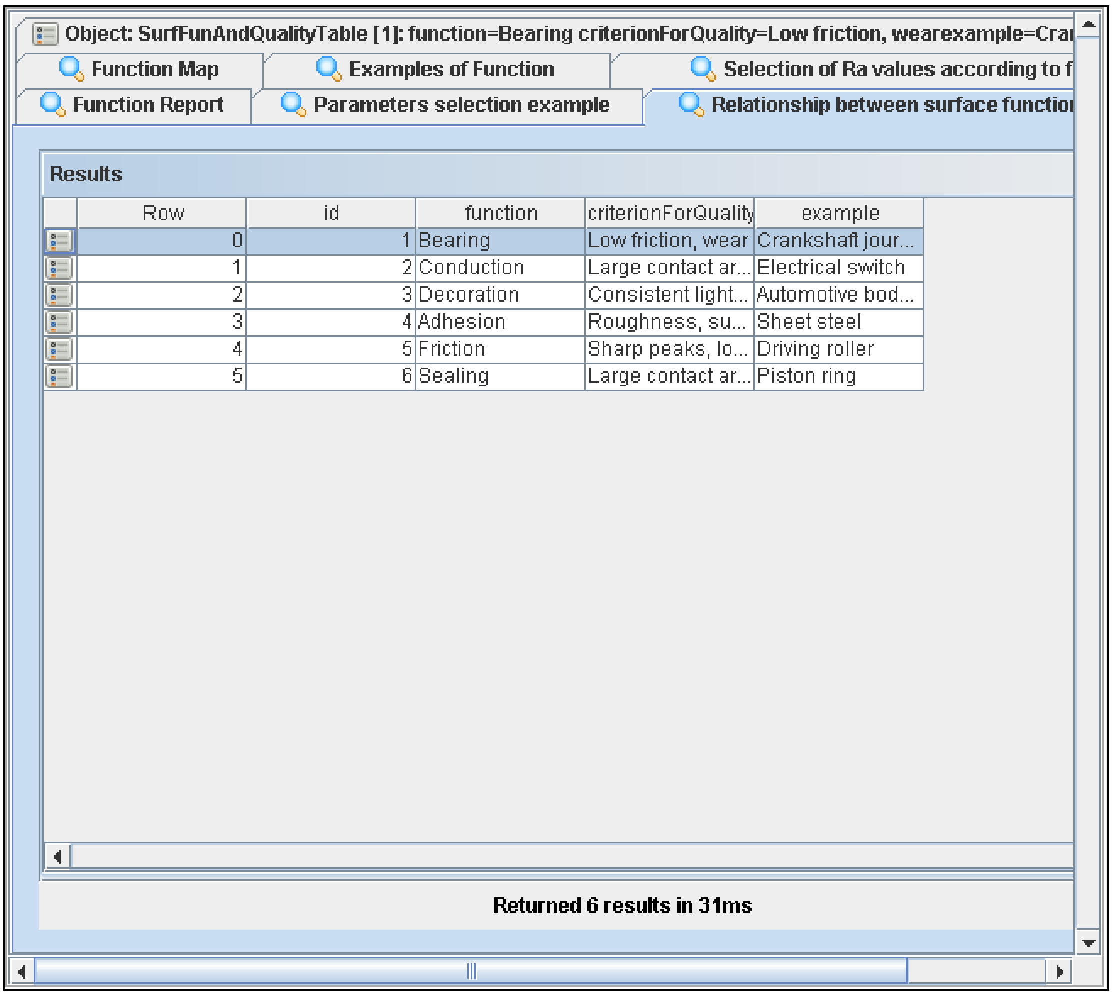
Function Map
Examples of Function
Selection of Ra values according to f
Function Report
Parameters selection example
Relationship between surface functio

Results

Row	id	function	example	Suggested_valu...
0	1	Non-functional	Engine block	50
1	2	Fluid friction	Ship hull plate	25
2	3	Joint face (soft ...	Pipe flange	12.5
3	4	Assembly face	Gearbox housing	6.3
4	5	Bearing(slow s...	Hand crank	3.2
5	6	Clearance fit	Bolt shank	1.6
6	7	Lubricated bear...	Gear teeth	0.8
7	8	Bearing(high sp...	Hand crank	3.2
8	9	Rolling bearing,...	Valve follower H...	0.2
9	10	Measuring refer...	Gauge	0.1
10	11	Performance se...	Fuel injector	0.05
11	12	Wring surface	Gauge block	0.0025

Returned 12 results in 32ms

APPENDIX J – RELATIONSHIP BETWEEN SURFACE FUNCTION AND QUALITY


Object: SurfFunAndQualityTable [1]: function=Bearing criterionForQuality=Low friction, wearexample=Cra
Function Map
Examples of Function
Selection of Ra values according to f
Function Report
Parameters selection example
Relationship between surface functio
Results

Row	id	function	criterionForQuality	example
0	1	Bearing	Low friction, wear	Crankshaft jour...
1	2	Conduction	Large contact ar...	Electrical switch
2	3	Decoration	Consistent light...	Automotive bod...
3	4	Adhesion	Roughness, su...	Sheet steel
4	5	Friction	Sharp peaks, lo...	Driving roller
5	6	Sealing	Large contact ar...	Piston ring

Returned 6 results in 31ms

APPENDIX K – A COMPARISON BETWEEN RELATIONAL DBMS AND THE CATEGORICAL DBMS

Relational Notions	Categorical Notions	Explanations
Field	Internal Object	A file in relational DBMS must be atomic (e.g. string or number) and always stored physically. An internal object of a category in categorical DBMS can be structured, e.g. they can be represented by other categories. Internal object can also be computed through a method.
Row/Record	Instance Category	Records must be formed by atomic data elements such as number, character or date. They can not contain other records as inner fields. However, instance category can contain other instance categories as internal objects of a category. Instance object categories are not always stored directly, but are computed dynamically by methods. Instance object categories generated by methods can also be stored in categorical database.
Definition of Row/Record	Class Category	As magnified by the "impedance mismatch" problem of relational DBMSs, a record definition does not directly map onto a type of a programming language. It must always be converted back or forth for this purpose. Class categories can directly describe the real world entities, so they are much easier to be understood by users. Moreover, class category can also encapsulate business logics (behaviors) together with the targeting data, keeping it all conveniently in one place. Class category can be implemented by the Java class directly. A Java class is a data definition of Java programming language.
Table	CTCollection	Tables and CTCollections are similar in both database notions as they both contain many records or instance categories. They both have indexing structures for faster access. However, tables have a very rigid structure (e.g. all rows in a table must have same definition, same fields). CTCollection can contain instance categories of different class categories.

Query	Functor+Filter	A relational query can only specify a table as the result, which means that all records have to be in the same type. The result is restricted to one dimension. The “filter” in a categorical DBMS is similar to the “WHERE” clause in relational query. The query strategy in the categorical DBMS is through functor mapping from category to another category with filters taken out unsatisfied instance categories.
Primary Key	Identifier	Primary keys are used to identify one record from others. Relational DBMS users are responsible for defining keys conforming to Normal Forms. In the categorical DBMS, a unique identifier is assigned to every instance category automatically based on the physical storage addresses. Therefore, users of the categorical DBMS can avoid the error prone process of defining keys.
Foreign Key	Product	In a relational DBMS, records from different tables are combined together using foreign keys. While this is a simple mechanism, it is quite slow, and hard to maintain. On the other hand, a categorical DBMS uses product construct to link different categories, which can link categories directly using Java object references mechanism. This is a faster process than the key lookups and can be maintained by the categorical DBMS automatically.
Join	Coproduct	Related records in a relational DBMSs are brought together using the “JOIN” operation. Joins are slow when more than a few tables are involved. For two tables (m × n) combinations, every extra table involved this figure has to be multiplied by the size of the table. Coproduct on the other hand is a very fast single step.

APPENDIX L – THE DATABASE SCHEMA DEFINITIONS FOR CALLOUTS IN THE P/FDM

```
create temporary module callout

declare feature ->> entity
declare feature(feature) -> string
key_of feature is feature

declare areal ->> feature

declare length ->> entity
declare length(length) -> float
key_of length is length

declare num_up ->> entity
declare num(num_up) -> integer
declare up(num_up) -> float
declare determines(num_up,float) -> length
key_of num_up is num,up

declare num_cutoff ->> entity
declare n_name(num_cutoff) -> string
declare num(num_cutoff) -> integer
declare up_(num_cutoff,integer) -> num_up
key_of num_cutoff is n_name

declare uplimit ->> entity
declare uplimit(uplimit) -> float
declare num_(uplimit,float) -> num_up
key_of uplimit is uplimit

declare lowlimit ->> entity
declare lowlimit(lowlimit) -> float
key_of lowlimit is lowlimit

declare bandwidth ->> entity
declare has_uplimit(bandwidth) -> uplimit
declare has_lowlimit(bandwidth) -> lowlimit
key_of bandwidth is key_of(has_uplimit)

declare f_type ->> entity
declare f_name(f_type) -> string
declare f_type(f_type) -> string
key_of f_type is f_name

declare filter ->> entity
declare has_bandwidth(filter) -> bandwidth
declare has_f_type(filter) -> f_type
key_of filter is key_of(has_bandwidth), key_of(has_f_type)

declare type_value ->> entity
declare types(type_value) -> string
declare range(type_value) -> integer
declare values(type_value) -> float
declare default_determines(type_value) -> bandwidth
declare default_determine(type_value,string) -> filter
key_of type_value is types, range
```

```
declare p_type ->> entity
declare p_type(p_type) -> string
declare value_(p_type,string) -> type_value
key_of p_type is p_type

declare name ->> entity
declare name(name) -> string
key_of name is name

declare parameter ->> entity
declare has_name(parameter) -> name
declare has_p_type(parameter) -> p_type
key_of parameter is key_of(has_name)

declare determine(name,string) -> p_type

declare value ->> entity
declare value(value) -> float
declare range(value) -> integer
declare type_(value,string) -> type_value
key_of value is range, value

declare t_type ->> entity
declare t_name(t_type) -> string
declare t_type(t_type) -> string
key_of t_type is t_name

declare tolerance ->> entity
declare has_parameter(tolerance) -> parameter
declare has_t_type(tolerance) -> t_type
declare has_value(tolerance) -> value
key_of tolerance is key_of(has_parameter), key_of(has_t_type),
key_of(has_value)

declare c_type ->> entity
declare c_name(c_type) -> string
declare c_type(c_type) -> string
key_of c_type is c_name

declare comparule ->> entity
declare has_c_type(comparule) -> c_type
declare has_num(comparule) -> num_cutoff
key_of comparule is key_of(has_c_type), key_of(has_num)

declare direction ->> entity
declare d_name(direction) -> string
declare direction(direction) -> string
key_of direction is d_name

declare profile ->> feature
declare has_filter(profile) ->> filter
declare has_length(profile) -> length
declare has_direction(profile) -> direction

declare sim_callout ->> entity
declare callout(sim_callout) -> string
declare has_feature(sim_callout,string) -> feature
declare has_tolerance(sim_callout,string) -> tolerance
declare has_comparule(sim_callout,string) -> comparule
```

```
key_of sim_callout is callout

declare callout ->> entity
declare callouts(callout) -> string
declare has_callouts(callout) ->> sim_callout
key_of callout is callouts;
```

APPENDIX M – THE POPULATING CODES FOR CALLOUTS IN THE P/FDM

```
entity direction;
d_name;direction;
*;
"";"";
"_|_";projection;
c;circular;
*;

entity f_type;
f_name;f_type;
*;
"";"";
Gaussian;Gaussian;
"2RC";"2RC";
*;

entity t_type;
t_name;t_type;
*;
"";u;
l;l;
*;

entity name;
name;;
*;
Ra;;
Rq;;
Rsq;;
Rku;;
R_q;;
Rz;;
Rv;;
Rp;;
Rc;;
Rt;;
Rsm;;
Wa;;
Pa;;
*;

entity p_type;
p_type;;
*;
aveamp;;
maxamp;;
spacing;;
*;

function determine;
name, string; p_type;
*;
[Ra], p; [aveamp];
[Rq], p; [aveamp];
[Rsq], p; [aveamp];
```

```
[Rku], p; [aveamp];
[R_q], p; [aveamp];
[Rz], p; [maxamp];
[Rv], p; [maxamp];
[Rp], p; [maxamp];
[Rc], p; [maxamp];
[Rt], p; [maxamp];
[Rsm], p; [spacing];
*;

entity value;
range,value;;
*;
4,3.3;;
*;

entity type_value;
types,range;;
*;
aveamp,1;;
aveamp,2;;
aveamp,3;;
aveamp,4;;
aveamp,4;;
aveamp,5;;
maxamp,1;;
maxamp,2;;
maxamp,3;;
maxamp,4;;
maxamp,5;;
spacing,1;;
spacing,2;;
spacing,3;;
spacing,4;;
spacing,5;;
*;

entity uplimit;
uplimit;;
*;
2.5;;
8.0;;
0.8;;
0.08;;
0.25;;
*;

entity lowlimit;
lowlimit;;
*;
0.0025;;
0.008;;
0.025;;
*;

entity bandwidth;
key_of(has_uplimit); key_of(has_lowlimit);
*;
[0.08];[0.0025];
[0.25];[0.0025];
```

```
[0.8];[0.0025];
[2.5];[0.008];
[8.0];[0.025];
*;

function default_determines;
type_value; bandwidth;
*;
[aveamp,1];[[0.08]];
[aveamp,2];[[0.25]];
[aveamp,3];[[0.8]];
[aveamp,4];[[2.5]];
[aveamp,5];[[8.0]];
[maxamp,1];[[0.08]];
[maxamp,2];[[0.25]];
[maxamp,3];[[0.8]];
[maxamp,4];[[2.5]];
[maxamp,5];[[8.0]];
[spacing,1];[[0.08]];
[spacing,2];[[0.25]];
[spacing,3];[[0.8]];
[spacing,4];[[2.5]];
[spacing,5];[[8.0]];
*;

entity num_cutoff;
n_name;num;
*;
"";5;
1;1;
2;2;
3;3;
4;4;
5;5;
6;6;
7;7;
8;8;
9;9;
*;

entity num_up;
num,up;;
*;
5,2.5;;
8,8.0;;
*;

entity length;
length;;
*;
12.5;;
64.0;;
*;

entity c_type;
c_name;c_type;
*;
"";"16%";
max;max;
*;
```