A distributed PDM system

Litjens, Vincent

Award date:
2000

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computing Science

MASTER'S THESIS

A Distributed PDM System

By
Vincent Litjens

Supervisor: dr. A.T.M. Aerts

Advisors: dr. ir. H.J. Pels
ir. F. Ploegmakers

June 2000

Preface

This report describes the activities carried out for my graduation project at Philips Display Components in the TPI Support Group of the PPD. The graduation project is the final phase of my study at the Department of Mathematics and Computing Science of the Technical University of Eindhoven. Within my study I have chosen to specialise in Information Systems, of which a PDM system is a variant.

This report is of value for those who are interested in PDM systems in practice. It gives a conceptual design for sharing product information between geographical different located sites with the use of a PDM system. The report is written in such a way that the reader doesn't need special knowledge of computing science.

I want to thank Frank Ploegmakers and the TPI Support Group for giving me the opportunity to accomplish my graduation project. During my graduation period I have spoken to many people of Philips Display Components, hereby I want thank them all for their advice and contributions to my graduation. Furthermore, I want to thank Ad Aerts and Henk Jan Pels of the Technical University of Eindhoven for their advice and guidance. Finally, I want to thank all the people who supported and helped me during my whole study.

Eindhoven, June 2000

Vincent Litjens

Summary

Philips Display Components designs and manufactures/assembles display tubes for televisions and graphics monitors. The company operates a worldwide product chain, producing tube-coil assemblies at several sites in Europe, Brazil, USA, Taiwan and China. Design and manufacturing tube-coils involves a lot of documentation. In order to manage and structure this load of information a PDM system called IRIS (Interactive Response Information System) is used. Product Data Management (PDM) is a term, which encompasses a wide range of data management techniques. PDM systems organise, monitor, control, manage and secure all of the data and processes related to the design, manufacture and support of products throughout their life cycles. During the graduation period at Display Components the PDM system IRIS is investigated to improve the way the data is exchanged between the sites.

The product information of the tube coil is stored in a product structure. The product structure is a tree graph, representing the physical breakdown of a product. The structure is hierarchical, where the levels (nodes in the tree) represent components and sub-components. Next to the physical build up of the tube coil, the tree graph shows how the product documentation (design and manufacturing) is related to each other and stores information of the documentation (e.g. author, status, last modified data, etc.). This information is called meta-data and it is stored in objects. One of the defined objects is of the type *Item*. An item represents the product or a component, the node of the tree. To an item, objects of the type *Dataset* can be attached. An object of the type dataset represents a file (e.g. Word document, a drawing, sheet, etc.); it stores the meta-data of a file. Furthermore, an item can have a Bill of Material (BOM), which contains the links to sub-components of the component in question. Taking the product tree, the BOM contains the links to the 'children' of a node. The complete build-up of a product in components can be reflected in a product tree. A PDM system manages this product structure. In this report, an analysis is made of the product structure in IRIS.

Components are designed and manufactured at multiple locations (sites) in the world. The PDM system IRIS manages the product information of each site, but the system is restricted to manage only the data of the sites independently. The PDM system IRIS is currently operating independently at eleven sites. There is no integration or communication between the PDM systems. Here lies the gap; the product information is not limited to one site, but needs to be exchanged between multiple sites. From this gap my graduation assignment arose. There has to be a PDM system with all its functionality, which covers all the sites and treats them as one global 'site', with a centralised view of all the data. *The graduation assignment is to investigate, starting from the current situation and the user requirements for distributing product information, the possibilities for distribution and make a conceptual design of a distributed PDM system.*

The main property of the distributed PDM system is that the product data is accessible for all the sites divided over the world and that the integrity of the data is maintained during modifications. The desired system has to make it possible to construct a distributed product structure. Parts of a product tree can be created at different sites and taken together to assemble one product tree. Ownership of the parts stays in hand of the creators (or transferred to the proper person). Users of different sites need to add data to the product structure and make this information accessible for other users.

The distribution facility, which IRIS offers at the moment, is not suitable to construct a shared product structure. In order to provide the functionality for a shared product structure, two architecture concepts are proposed. In the architectures, storage of meta-data and storage of files are taken separately. The files have to be stored close to the user, the size of the files is too large to transport them frequently over the global network. Therefore in both concepts the files will be stored locally at each site. For meta-data two concepts are worked out; meta-data stored in one central database on one site and meta-data stored at the same place as the files, resulting in

a distributed database system. A distributed system contains independent operating databases, located at every site, communicating with each other in order to form *one logical* database. Both architectures can provide the same functionality, but have different advantages and disadvantages. The advantage of a central database is that all meta-data is stored in one location. This is easier to manage and to maintain, and it makes the system less complex. The main disadvantage is that all requests of clients will be transmitted over the global network. This demands a lot of network capability. The network load will be much lower with a distributed database. The principle of a distributed database system is that the meta-data will be replicated among the sites. The request of the client can be handled locally, because the meta-data is stored locally too. Global transactions only occur when data is modified (updating the replicas). It can be assumed that data will be much more read than modified. Therefore the network requirements are much lower. Besides, a distributed database system offers additional functionality, like local autonomy. The disadvantage of the system is that it is more complex and more difficult to configure. Furthermore the object model of the product structure has to be changed, so that it becomes possible to relate data of multiple sites. The principle behind relating the data is to introduce *relationship* objects, which connects one *data* object with the other. The data objects themselves will not be changed, but a new *relation* object will be created to link data objects together. It doesn't matter whom or which site owns the data object.

Both architecture concepts lead to the desired functionality. For both concepts the current PDM system needs to be altered. The PDM system IRIS is a variant of the PDM system iMAN of the vendor Unigraphics Solutions. IRIS is a turnkey product delivered to Display Components. Display Components can defines it's own business rules and determines itself how the PDM philosophy will be implemented. The design and the 'production' of the system are still in the hands of UG Solutions. It still needs to be investigated, which concept is the easiest to implement.

In this report, a description of PDM is given, together with an impression of the PDM system IRIS. An analysis is made of the product structure stored in IRIS and some improvements to the model are given. Next, the basic requirements for a distributed product structure are investigated and compared with the functionality of the current distribution facility of IRIS. It is concluded that the current PDM system doesn't provide the functionality for distribution, which is needed. Subsequently, two architectures are proposed, and distribution aspects are explained. This report will give conceptual solutions for a distributed product structure and pinpoints the aspects of distribution, which have to be taken into consideration when a 'new' system will be designed.

A Distributed PDM System

Content

1. Introduction

This report covers the activities carried out for my graduation project. My graduation project consists of investigating the requirements for distributing product information and to make a conceptual design of a distributed PDM system. The primary requirement of the PDM system is that the product information stored in the system is globally accessible. The first chapter contains a brief description of Philips Display Components, the company, where this graduation assignment took place. Furthermore, short explanations of the related terms of product data management (PDM) are given followed by a brief description of my graduation assignment.

In chapter 2, an explanation of what PDM comprehends and an introduction of the PDM system IRIS is given. The next chapter is devoted to the data, which is stored in the PDM system. An analysis of the data model is made in chapter 3. Chapter 4 describes the problem area and gives more details of the graduation assignment. Furthermore, the requirements of the desired system are specified. Chapter 5 explains the current distribution facility of the PDM system and shows the limitations of this facility (ODS). In chapter 6, possible architecture concepts are introduced and compared in order to establish the requirements of the desired system. A distinction between meta-data and physical data is made and for both, outlines of solutions for distribution and sharing the data are given. In chapter 7 the concept of a distributed database system, forming one logical database, is worked out in detail. In the last chapter recommendations for future development of the system IRIS are given.

1.1. Philips Display Components

Philips Display Components designs and manufactures/assembles display tubes for televisions and graphics monitors, both for Philips and other customers. The company operates a worldwide product operation, producing tube-coil assemblies at several sites in Europe, Brazil, USA, Taiwan and China. Products are made in one or more factories and product lines can be transferred between factories during the life cycle of the product. The tube components can be made in-house, elsewhere in Philips or from outside suppliers. Tube components and materials may also be supplied to third party customers. Finished tubes assemblies go to other Philips factories and external customers for integration into finished televisions and monitor products.

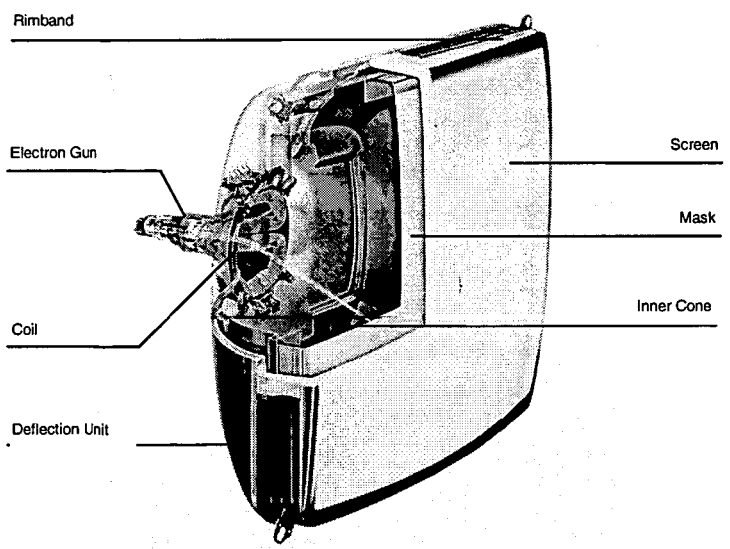


fig1.1: Tube Coil

Although the tubes are relatively simple in terms of the number of parts (hundreds rather than thousands), the product processes involved (coating, heat treatment, sealing, etc.) are numerous, complex and contain much documentation.

Philips Display Components headquarters is situated at Eindhoven, where the Product and Process Development group (PPD) is also situated. The PPD is the central development organisation for products and production processes of Display Components. TPI Support Group is a department of the PPD. TPI Support Group has as service the providing of support towards the implementation and use of PDM (Product Data Management) and management of the PDM system called IRIS (Interactive Response Information System). IRIS can be best viewed as a design system with management facilities for the whole tube-product design. The IRIS system is a tool, enabling the Display Components to develop efficiently and to use TPI (Technical Product & Process Information). The TPI system facilitates the *communication* about development and production of product, process and equipment between all parties concerned. It's main objective is to present the right information instantly to all those who need it.

1.2. Concurrent Engineering

To compete with the rivals, companies try to bring their products faster to the market. A method to reduce the time-to-market is concurrent engineering. The method is a communication process whereby all phases between development and manufacturing are integrated. With sequential engineering, the development phase will first be completed and then the manufacturing phase will start. There is no interaction between the two phases. With concurrent engineering the manufacturing phase already starts before the development phase is finished. Close interaction between the two phases is necessary, because on this way improvements of the product can be made much earlier than with the conventional engineering methods. The demands to make concurrent engineering possible are communication and information exchange. A PDM system answers to the demands of a concurrent engineering environment. Figure 1.2 illustrates the difference between sequential and concurrent engineering.

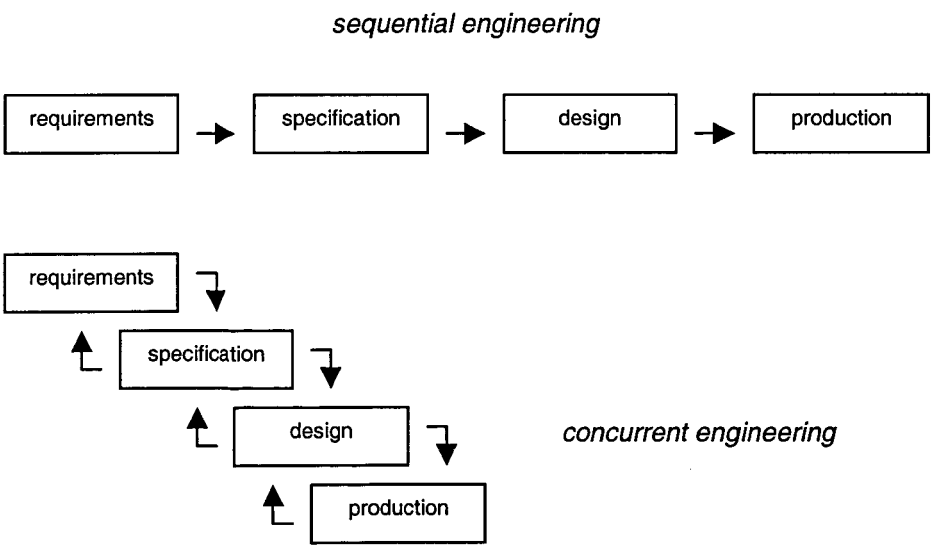


fig.1.2: Engineering methods

1.3. Product Data Management

Product Data Management (PDM) is a term, which encompasses a wide range of data management techniques. Amongst these are Engineering Data Management, document management, product information management and image management. PDM systems organise, monitor, control and manage all of the data and processes related to design, manufacturing and support of products throughout their life cycles. PDM integrates and manages processes, applications and the information that defines products across multiple systems and media. PDM manages this through well-organised data structures, permitting close co-ordination with concurrent engineering techniques, CAD/CAM, MRP and other manufacturing information. Essentially, PDM is an enabling technology for concurrent engineering. More explanation of PDM is given in chapter 2.

1.4. Product & Process Structures

In order for a complex object such as a tube coil assembly to be efficiently designed, and manufactured world-wide, the assembly and the information about the manufacturing of the assembly is divided, usually hierarchically into smaller and smaller process steps, objects or parts. Complex objects need to be described from different points of view in order to give efficient access to the relevant information on processing, assembly and design. For tube coil assemblies, the physical build-up (product structure) and the detailed process steps (process structure) can be charted in structures (fig.1.3).

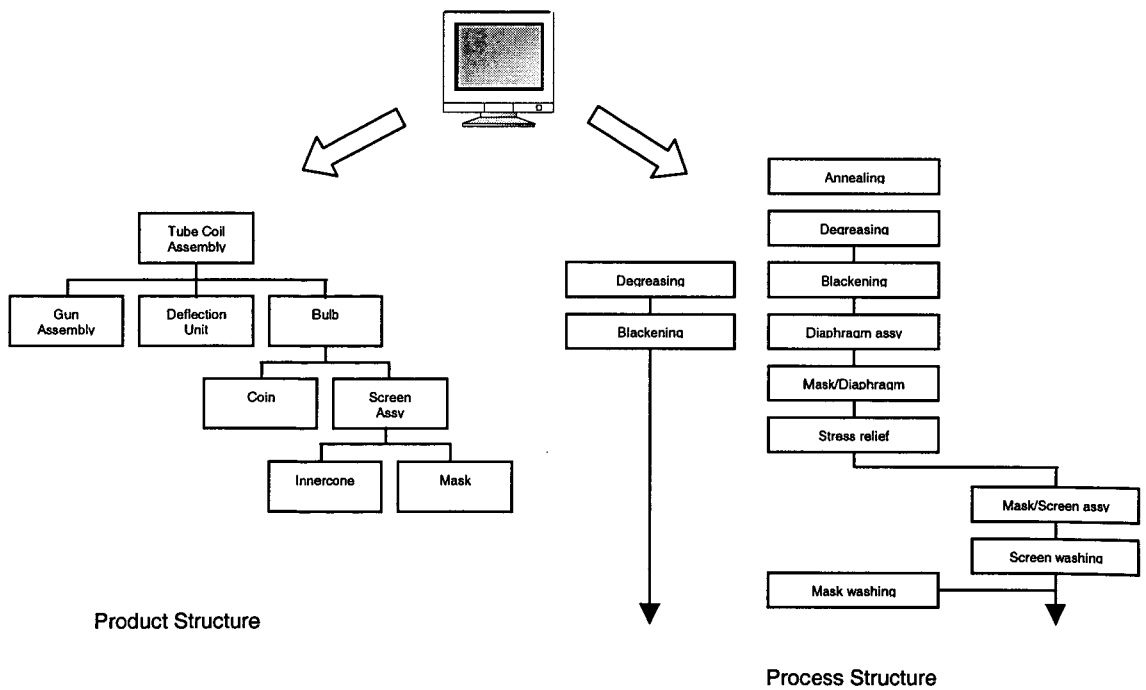


fig.1.3: Different structural views

A PDM system supports multiple structures for products (fig. 1.3). Product structures represent the physical product breakdown. The structure is hierarchical where the levels represent components, sub-assemblies and single parts. The process structure models the process steps, which have to be taken to manufacture the product. Each step contains input (components,

materials, chemicals, etc), related information (process descriptions, work instructions, etc.), equipment and chemicals which are used in the process and as output processed components, assemblies, waste, etc. The product structure describes the product itself, the requirements, specifications, etc. The process structure describes how to make the product, for example which temperature is needed to burn the glass. Information in both structures is related and references can be made from one structure to another and visa versa. Detailed information of the product structure used by Display Components is given in chapter 3.

1.5. Information Flow

TPI (Technical Product and Process Information) consists of all technical information that is needed to develop and manufacture a specific product. The flow of TPI of a tube coil can split up in two basic stages, design and manufactory. Currently the main design activities take place in Eindhoven, manufactory is dispersed over the world. During the design stage, all the involved groups want to keep track of the design process and want to be able to access the created documentation (TPI). In this way preparations for manufactory can be taken and adjustments of the manufactories to the design can be given in an earlier stage. It is imaginable that in the future the design stage will also take place at geographic separated locations. Next to this the manufacturing information has to be available for the design, so that communication about the product between all the parties can be established.

The PDM system IRIS is the tool, which manages the product information and the information flow.

1.6. Assignment Description I

Currently, multiple installations of a PDM system, called IRIS, are in operation as independent systems at various sites. Facilities for distribution of product data between the sites are available, but these do not offer the desired functionality.

Assignment:
Investigate, starting from the current situation and the user requirements for distributing product information, the possibilities for distribution and make a conceptual design of a distributed PDM system.

The focus will be on the product data and not on the process data, because a large part of the process data is local and dependable of the site. Besides the distribution of the global process data is not clearly defined at the moment. More information about the product information flow and the assignment is given in chapter 4.

2. Product Data Management (PDM)

This chapter explains the term Product Data Management. In the first paragraph the background of PDM is described, followed by a paragraph explaining the functionality and the benefits of a PDM system. In the last paragraph an impression of the PDM system IRIS is given.

Over the last several years, companies have increasingly invested in technologies and approaches to improve their ability to deliver innovative products to market, and thus improve their ability to compete in a global market. Forward thinking companies, like Philips Display Components are investing in Product Data Management (PDM) in order to better manage their product information. The challenge is to maximise the time-to-market benefits of concurrent engineering while maintaining control of the data and distributing it automatically to the people whom need it.

2.1. History

PDM systems initially grew out of the need to manage computer-aided design (CAD) related data. They were first developed to manage and control the volumes of electronic media that were created by engineering design automation. Manufacturing companies are usually good at systematically recording component and assembly drawings, but often they do not keep comprehensive records of attributes such as 'size', 'weight', 'where used' etc. As a result, engineers often have problems accessing the information they need. This leaves an unfortunate gap in their ability to manage their product data effectively. Data management systems should be able to manage both attribute and documentary product data, as well as relationships between them, through a relational database system. With so much data being generated, a technique to classify this information easily and quickly needed to be established. Access and viewing tools were developed along with data vaulting technologies to store and protect data from unauthorised use or modification. In addition, PDM capabilities were expanded to support the design process by electronically routing documentation through the review cycle. This paperless workflow reduced the duration of the product development cycle. Data could be extracted to trace the histories of every product and component with its relevant documentation. Soon it became possible to extend these benefits beyond the design and engineering departments into purchasing, manufacturing, and product support. This allowed an organisation to synchronise all product functions during the entire life cycle and to ensure the tracebililty of product histories.

2.2. PDM Systems

PDM systems and methods provide a structure in which all types of information used to develop, manufacture and support products are stored, managed, and controlled. Typically, Product Data Management (also called Engineering Data Management or Engineering Database) will be used to work with electronic documents, digital files, and database records, like engineering drawings, text documents, product specifications, analysis results, sheet and many others. The ideal PDM system manages product data throughout the enterprise, ensuring that the right information is available for the right person at the right time and in the right form. In this way, PDM improves communication and co-operation between diverse groups and forms the basis for organisations to restructure their product development processes and fits in exactly to the demands for concurrent engineering. In brief, any information needed throughout a product's life can be managed by a PDM system, making correct data accessible to all people and systems that require it. PDM is not limited to control only the consolidated product data and to manage the engineering change processes, but can manage product conception, detailed design, prototyping and testing as well. The product development *process* is managed as well as the *data*. The implementation of PDM

technology does not have to be an all-or-nothing undertaking. PDM will provide significant productivity gains when it is used by a working group, e.g. a product development team. PDM systems encourage teams to share data and documents. By doing so, data inconsistencies, use of out-of-date data, and dissemination of uncontrolled copies of documents can be avoided. As such, PDM systems serve as enablers for implementing concurrent engineering practices.

PDM definition:
PDM systems organise, monitor, control and manage all of the data and processes related to the design, manufacture and support of products throughout their life cycles.[1]

PDM functions:
PDM is a technology, which is still under development. While the technological base is growing rapidly, there are no standards till now, that defines the functionality necessary to have useful PDM systems. The definitions that follow are based on CIMdata's extensive consulting engagement with PDM users and suppliers.

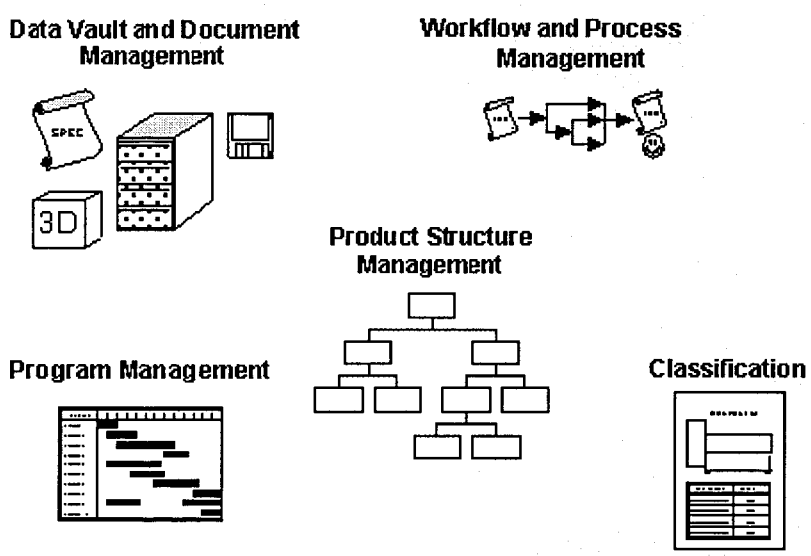


Fig.: PDM functions

Data Vault and Document Management

Data Vault and Document Management provides secure storage and retrieval of product data. This function ensures that this data is up to date, correct, and protected from accidental or deliberate damage. The vault contains either the data itself or information that points at the actual location of the data, e.g. CAD files. Data controlled by PDM cannot be accessed without going through the PDM system's control procedures, i.e. their check-in and check-out functionality. The location of the data is essentially hidden from users and applications.

[1] def. PDM : www.CIMdata.com. CIMdata provides worldwide strategic consulting and in-depth research for industrial organizations and suppliers of technologies and services seeking competitive advantage in the global economy., and within the evolving e-business world. The company helps organizations improve their development, use, implementation, and marketing of solutions that manage the entire product or plant definition lifecycle.

The following aspects are considered under Data Vault and Document Management:

- secure data storage (vault)
- check-in and check-out
- access control (authorisation, status)
- viewing, printing, plotting
- document structure management (hierarchy, view, variant)
- version control
- distribution control

Workflow and Process Management

Using Data Vault and Document Management alone, a PDM system can react to user's ad hoc demands. With Workflow and Process Management, a PDM system can, in addition, be proactive. It can interact with people according to predefined processes, and repetitive processes can be programmed within the PDM system. The main workflow management applications within PDM are: review and approval cycles, release procedures and engineering change procedures. The workflows are defined in terms of a sequence of events that must occur before the modified product data are allowed to get a certain status. Under Workflow and Process Management the following function can be considered:

- review procedure support (mark-up & red lining)
- approval procedure support (sign-off)
- release procedure support (status change)
- problem reporting support procedure
- engineering change support procedure
- support of company defined procedures
- dynamic routing of documents
- tracking of processes / history management

Product Structure Management

Product Structure Management facilitates the creation and management of product configurations and Bills of Material. The product data can be accessed by product structure. For any selected product, the relationship between its component assemblies and between the parts that make up these assemblies can be maintained. Complete Bill of Material (BOM) can be opened, including documents and parts, either for the entire product or selected assemblies. PDM systems allow users and applications to link product definition data such as drawings, specifications and other documents, to the parts and assemblies in the product structure. As such, the product structure can be regarded as the backbone for structuring all information comprised in the PDM vault. Product structure information is common to PDM systems, CAD systems ("assembly modelling") and MRP systems (also called Business Management systems or Logistic systems). Product Structure Management includes the following functions:

- searching on product structure
- management of BOM structures
- management of variant structures
- version/revision control / change history
- status control
- report functions

Classification

The classification of objects allows similar or standard parts, processes and other design information to be grouped by common attributes and retrieved for use in products. This supports product standardisation, and leads to reduced re-design, savings in purchasing and production and reduced inventories. PDM classification functions provide much more efficient mechanisms for finding and selecting standard and preferred parts compared with catalogues and other manual systems. Search and maintenance functions can be based in two groups:

- search and maintenance functions based on hierarchical grouping of product classes
- search and maintenance functions based on attributes

Program Management

Program Management provides work breakdown structures and allows resource scheduling and project tracking. These capabilities are frequently provided through links to third party project management systems. A key advantage stems from the ability to relate the work breakdown structure tasks to the PDM system's knowledge of approval cycles and product configurations. The following aspects can be considered under Program Management.

- progress control
- interface to project management system

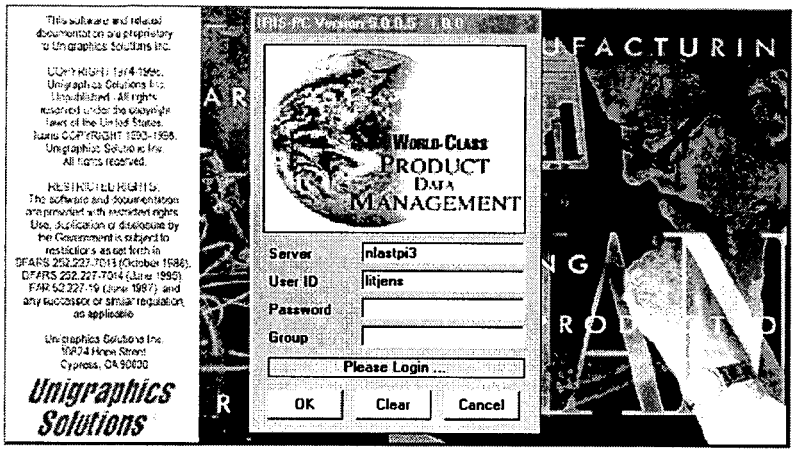
Utilities and Features

Other PDM functions include basic utilities for backup and recovery of data, system failure protection and on-line help. A set of utilities dealing with notification and communication is usually part of a PDM system. This set includes import/export functions, data translation services, and electronic mail. Furthermore, facilities to customise or tailor the PDM system are essential. These may include tools to adapt information models, tools to encapsulate or integrate applications and reporting facilities. Usually, PDM Systems that are based on electronic document management systems provide extensive image services.

A successful PDM implementation can bring many benefits to a company such as reducing engineering design time, increasing reuse of existing design solutions, cost savings in manufacturing, reduce time-to-market and increased product quality. In a complex enterprise-wide organisational structure like BG Display Components it is essential for all project members to get the information they require, in the right format, at the right time and to be aware of any change which may effect their work.
The implementation of a PDM system doesn't occur at once. It takes a long period of integration the system into the way of working of the company. Business rules have to be defined and put into the system. The benefits will not occur at once, but can be expected in steps during the implementation process.

2.3. The PDM System IRIS

The PDM system IRIS is a customised version of IMAN[2] by order of Philips Display Components. Information Manager (IMAN) is an object database management system. In this paragraph, an impression of IRIS is given. The main functionality is described and the software architecture of the system is briefly explained.



2.3.1 IRIS Functionality

In this paragraph a general outline of the IRIS functionality is described in order to give an impression how IRIS look like. The components of IRIS are briefly described; detailed functions are left out. IRIS has 3 main utilities; Workspace (WS), Product Structure Editor (PSE), Enterprise Process Modelling (EPM), which covers the following general functionality of a PDM system described in paragraph 2.2:

- Workspace -> Data Vault and Document Management
- Product Structure Editor -> Product Structure Management
- Enterprise Process Modelling -> Workflow and Process Management

Workspace

The workspace is a graphical desktop that displays the product information as various graphical *objects*. The workspace is a window containing two basic areas, an object area and a property area. The object area displays the product information as objects. The property area is used to display the attributes of the objects. Object folders are used to organise the various kinds of product information. These folders can be nested inside one another so that product information can be organised in the way that best meets the individual needs. (The objects are described in chapter 3.)

[2] The commercial PDM System IMAN produced by Unigraphics Solutions.

The following figure 2.1 shows a typical Workspace environment on a PC as it appears to user. On the left side is the object area, showing the folder structure and on the right side, the properties of the folders and objects are displayed. Each user can configure his/her own workspace by creating own folders/objects or making links to other folders/objects under folder *Home*.

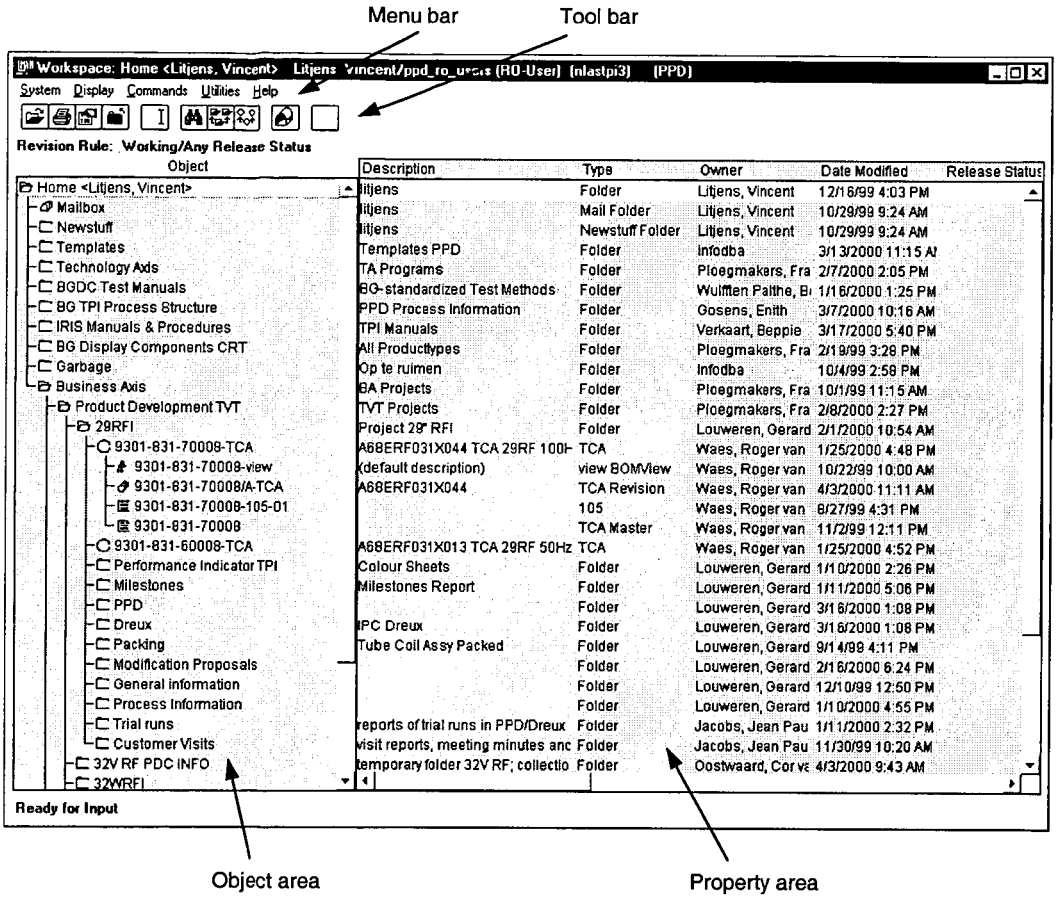


fig.2.1: Workspace IRIS-PC

Product Structure Editor

A graphical user interface, the Product Structure Editor (PSE) creates and modifies the product structure and its associated data (e.g. sequence number, assembly note, variant conditions for configuration). PSE displays the structure in a Bill Of Material (BOM) format very similar to those used in many CAD and MRP systems.
Structure can be created using simple *cut* & *paste* operations from Workspace or PSE or IDs and data can be typed in directly. The PSE display has facilities to help with browsing large BOMs and rapidly locating components within a BOM. PSM not only manages the geometric definition for a part or assembly, but it also handles all the associated data such as test result, supplier specifications, engineering notes and manufacturing information. A "where used" functionality is built into PSM, so the designer can check were elsewhere the component or data is used in other bills of material.

Figure 2.2 shows the PSE of IRIS-PC. On the left side the Bill of Material is shown, on the right side the properties of the components in the BOM are visual. The Bill of Material can also be

represented in a horizontal or vertical tree view. The user can navigate around in the left side of the window and find the wanted components. From any point in the structure it is possible to access the associated data. The PSE has also some additional utilities, like to format reports that can be printed.

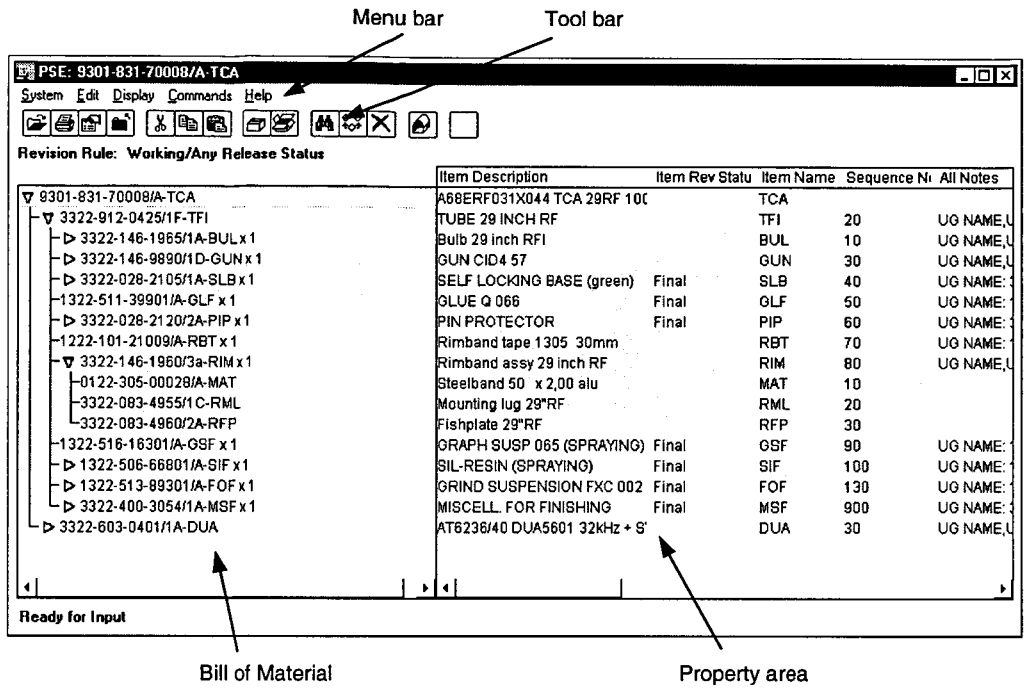


fig.2.2: Product Structure Editor IRIS-PC

Enterprise Process Modelling

Workflow

Workflow is the concept that in the real world all work progresses or flows through one or more processes in order to accomplish a goal or objective. Workflow involves interaction with the members of the working group electronically. Workflow processes require resources, like capital, raw materials, time and personnel. Moreover, in order to accomplish workflow objectives, all enterprises use *business rules*. The rules establish *conditions* for workflow processes. For example in order to accomplish a product design objective, certain conditions must be satisfied (e.g., the design must be completed in a timely manner). Enterprise Process Modelling (EPM) is used to model workflow processes, allocate resources and manage data according to business rules. Modelling of workflow processes is handled by procedures. A procedure is like a blueprint of a workflow process. A procedure describes the individual tasks and the task sequence required modelling the workflow process. Each task defines a set of actions, rules and resources used to accomplish that task and every task is always in one of the defined states. Task actions and states are closely connected concepts. Actions transfer a task from one state to another and the ultimate goal of every task is to reach the final state. Task-states are used to control and co-ordinate execution of each individual task in a procedure. Examples of procedure applications are the following:

Cascade Release

Cascade Release is used to create procedures for releasing objects. A Cascade Release procedure uses well-defined templates to automatically notify selected users requesting work

signoff. Typically, enterprises make a distinction between development data and released data. This is because development data may be changed many times before the design is approved. However, once a design is finalised, any changes made to released data are usually tightly controlled. A release procedure is a formalised process of announcing to the enterprise that development data is finalised (released). A release procedure can model a specific sequential approval process with multiple release levels. Each release level can contain multiple parallel signoffs. It is a template which specifies the number of signoffs required at each release level, as well as the user group and roles required to perform a sign-off (fig.2.3). At the end of a release procedure the released data has received a status (Quality level e.g. provisional, consolidated, etc.), with associated protection settings.

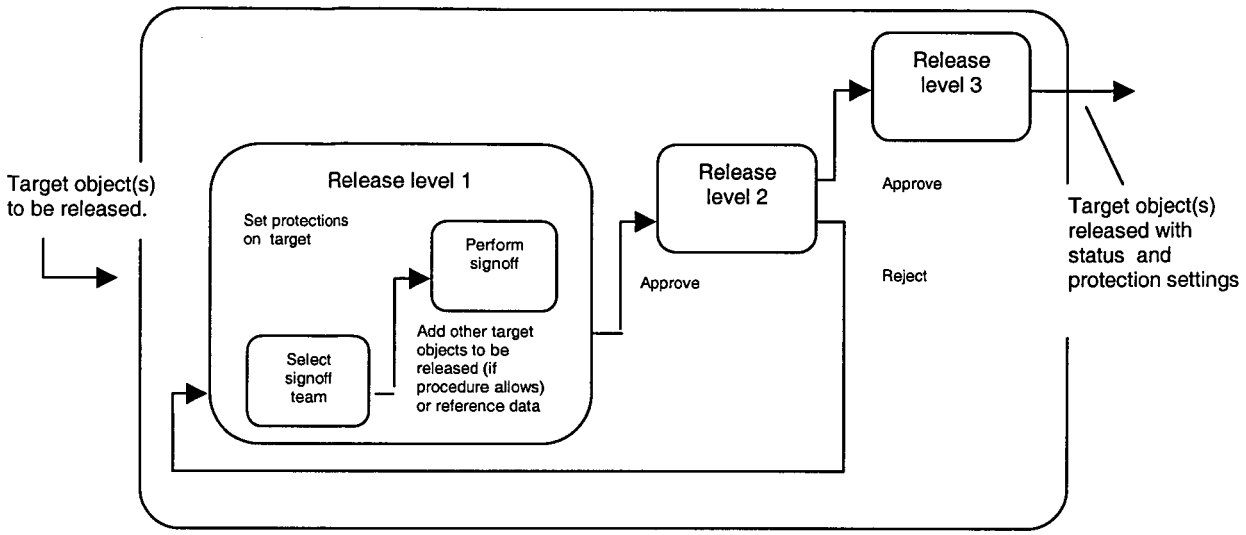


fig.2.3: Cascade procedure task flow diagram

Change Management
Change Management is intended to complement Cascade Release functionality by providing a formalised process for changing released data. In fact, a Change Management procedure must incorporate a release procedure in order to finalise those changes and release the changes to the enterprise. A Change Management procedure models the process for proposing, controlling and approving changes to a set of objects. Change Management is the structured decision process that takes care of the changes to information necessary for the business, following a pre-defined procedure.

The Change Management procedure serves two basic needs of the organisation:

- It enables the organisation to review the consequences of a contemplated change, prior to the decision to make the change.
- It ensures that all information is updated (and relevant people are informed) once it is decided that the proposed change will be implemented.

Before anyone can change information that is used throughout the enterprise, one needs to check the consequences of this change with everyone that is depending on the information (for their business processes). All people involved may then decide (more or less together) if the change will be acceptable and/or profitable. After this decision, the changes to the information can be implemented and all users of the information are notified about the change.

See *Appendix III: Release & Change Procedures* for more information

2.3.2 IRIS Software Architecture

IRIS is an object database management system, which is built on top on a relational database management system of the vendor Oracle. Instead of dealing with data records, the object database stores *data objects*. The product data structures are built up with objects, these objects are stored in a database.

IRIS is constructed in layers (fig.2.4). The lowest layers are the closest to the host Operating System (OS) facilities and the highest layers are the closest to the user. The IRIS architecture consists of:

- User Interface (UIF) layer
- Application and Utilities layer
- Core Modules layer
- System Modules layer

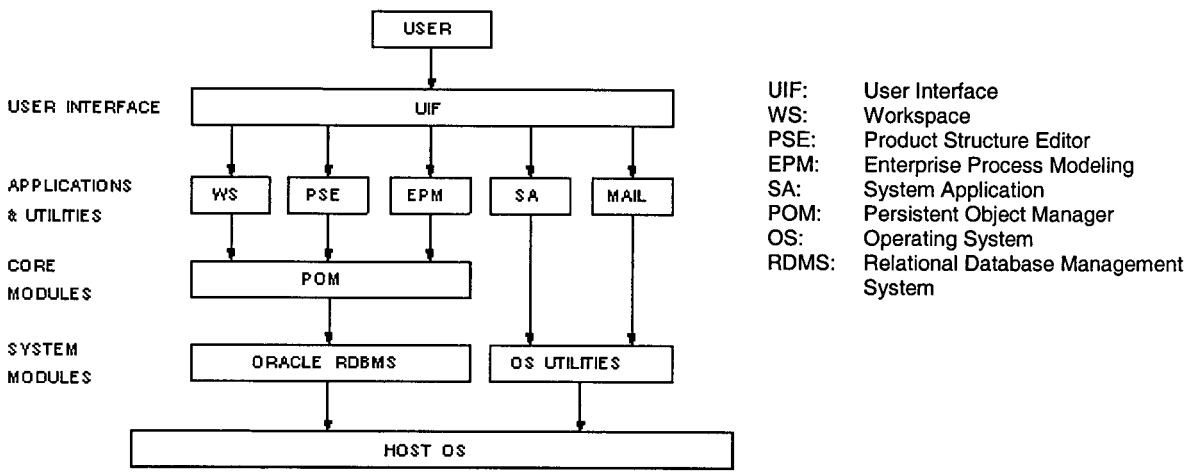


fig.2.4: IRIS Architecture

User Interface Layer
The user interface layer consists of software tools used to interact with the user (e.g., displaying information to the user and receiving data inputs and commands from the user.

Application and Utilities Layer
The Application and Utilities layer consists of modules that allow the user to view and perform work on the data objects. The application layer consists of modules such as the Workspace, Enterprise Process Modelling (EPM) and Product Structure Editor (PSE) or third-party applications such as Unigraphics, other CAD/CAM systems or text editors which are integrated in the system. Utility modules provide application functions such as the System Administration and Mail.

Core Modules Layer
The layer below the applications and utilities consists of core modules. These modules perform medium-level functions required by the application and utilities modules. The POM (Persistent Object Manager) provides the interface between the objects and the Relational Database Management System (RDBMS). The Persistent Object Manager takes cares of the movement between memory and the secondary storage. POM allows applications and utilities to interface to

the RDBMS at a higher level of data abstraction than relational tables (fig.2.5). For example, an application might request that the POM create a folder. The POM would interpret this command and generate the SQL statements necessary to create the necessary relational data.

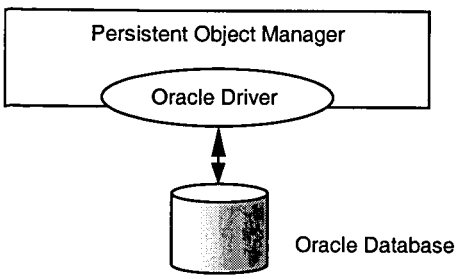


fig.2.5: Interface POM with Oracle

The POM provides the following functionality:

- Data definition services: Definition of classes by inheritance from existing classes and definition of attributes.
- Data manipulation services: Manipulation of in-memory objects and support for their saving and retrieval to and from the underlying RDBMS.
- Locking: Support for many different applications accessing the same data concurrently.
- Referential Integrity: Protection against the deletion of data used by more than one application.
- Access controls: Support for the access control lists attributed to objects. The access controls themselves are manipulated by functions provided by the Access Manager.

System Modules Layer
System modules interface core modules and some applications and utilities to the Oracle Database system and host operating system utilities. System modules include interfaces to the operating system for functions such as file opening, closing and saving, network access and inter-process communication.

In the next chapter the object model managed by the POM will be explained.

3. Product Structure

In this chapter, the product data stored and managed by the PDM system IRIS is described. The logical structure of how the company defines the product assembly. In the first paragraph the objects of the product structure are explained. The second paragraph shows the relations between the objects. In the last paragraph some comments about to the data model are given.

Product structures represent the physical breakdown of a product. The structure is hierarchical, where the levels represent components and sub-components. The basic structure of a Tube Coil Assembly (TCA) is in general always the same. The main differences lies in the details within the components. The assembly can be shown in a tree graph. The tree shows how the whole tube is constructed out of sub-constructions and basic elements. The root of the tree graph specifies the final product. Underlying nodes of the tree represent sub-components or single parts. There is always one unique tree for every tube design. There can't be two different trees of one tube type. In figure 3.1 the basic structure of a tube coil is shown. The product tree can be divided in two levels; assembly level and component level. Assembly level is the tree structure, the physical build up of a product (the bills of material). Component level is the content of each node in the tree, the description of a component.

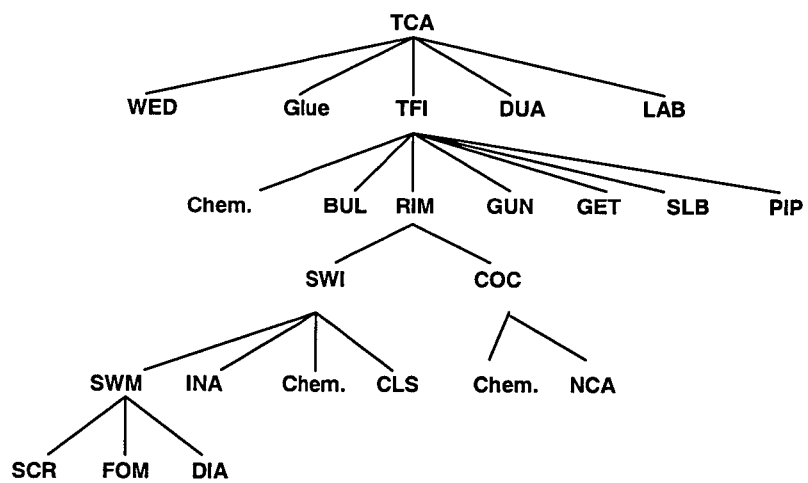


fig. 3.1: basic structure Tube Coil Assembly

In additional to the development view of a product, there will also be a manufacturing view, which can be different from the product structure. The manufacturing structure reflects the product items and assemblies, which are required during manufacturing. The conversion of the development view of a product to the manufacturing view is not clear defined yet. At the moment the development view is also used to store manufacturing information.

3.1. The Objects of the Product Structure

The data model of the product structure is built up with objects. In this paragraph the object types are described. The most important object types are *Item* and *Dataset*; an object of the type Item represents a product or component, an object of the type Dataset represents a file (e.g. Word document).

In this report an item will represent a product component. For process information a similar tree is used for storage, but the contents of a node in a process tree can be different than a product tree.

The logical structure of 'a process tree' is still under construction and is not clearly defined yet, but similar objects will be used. In this report the objects are used to store product data.

A node in the product tree is called an *item* in IRIS. *Items* are the fundamental objects used to manage information. They provide an identification for physical or conceptual entities about which an organisation maintains information. They represent something physical, which exist in reality, like the electron gun of a tube coil. Or they can represent something conceptual, like a process to make the chemicals for the tube coil. The product structure contains the following objects:

Item
An Item is used as a stepping stone for a product component, where all information of the particular component can be connected.
An Item has the following basic information:
- ID: A unique identifier for an Item (12NC).
- Name: A user defined name to describe the Item
- Type: A classification of Items that allow different types of Items to be treated separately. In the context of this paragraph an item represents a product component.
- Description: A text field to describe the Item.
- Standard attributes for ownership, date, last modifying user, etc.

Form
Next to the basic attributes, an Item can have additional attributes. This collection of additional attributes is stored in an object called *Form*. Object Form can be seen as a record where the user can define its own attributes (record fields) for describing an *Item*. (For example in IRIS one of these attributes is 'Supplier').

Dataset
A file, (a drawing, a document, a sheet, etc.) is represented by a *Dataset* in IRIS. A *Dataset* holds the meta-data for an actual data-file and contains the address to the physical data-file. Further a *Dataset* has the property to keep track of all the version of particular file. If the file is changed then the old version is still accessible. For the revised file, a new object of type Dataset is created with a version number one higher than the old version.

A *Dataset* has the following basic information:
- Name: A user defined name to describe the Dataset (is not unique)
- Type: A classification of the Dataset
- Tool Used: The tool that created the file, which the Dataset represents.
- Version: version-number of the file which the Dataset represents.
- Description: A text field to describe the Dataset.
- Standard attributes for ownership, date, last modifying user, etc.

Item-Revision
An *Item-Revision* is, like the name already indicates, an object, which is used to reflect modifications to an Item (a revision). This is not the same as the version control property of datasets. An *Item-Revision* depends on the kind of modification, for example if the specification of an item (product) is changed. This will not mean that there will be automatically a new revision, when version of a dataset attached to the particular item is changed. Usually, the last revision is the youngest, but not necessarily the one used for actual production at a given time. Each site can define its own procedures to determine how and when a new *Item-Revision* should be created. An *Item-Revision* is always connected to its' original item (parent item). There can't exist a stand-alone *Item-Revision*.

See appendix IV for Item Revision Rule

BOM
A *BOM* (Bill Of Material) is the IRIS object that is capable of holding structures. It is connected to an Item (-revision) and holds references to the Items that are the components of the Item in question (e.g. a product-Item representing an assembly has a *BOM* representing the components - also Product-Items – of that assembly). A *BOM* exists only if it is connected to an item (or item-revision). However an Item doesn't always have to contain a *BOM* (the leaves in the tree). A *BOM* holds a one-level structure, only identifying the Items that are one level down in the structure. It can be seen as the edges (references) to the children of the node in the product tree.

Folder
A *Folder* is a generic container of references to any object. Folders are a mean to structure information and thus facilitating the navigation through the information. Normally they are used to group related information. Object Folder is not included in the product structure, but product structures are reachable through a folder. The folder contains links to the product structures. Object Folder is not unique, it is possible to have multiple folders of the same name.

3.2. Relations between Objects

An object of the type item itself doesn't say much about a component. Contents can be given by adding other non-item objects to the item. The objects, which can be added to an item (node of the tree) are shown in the figure 3.2 below (see Appendix I for explanation object modelling).

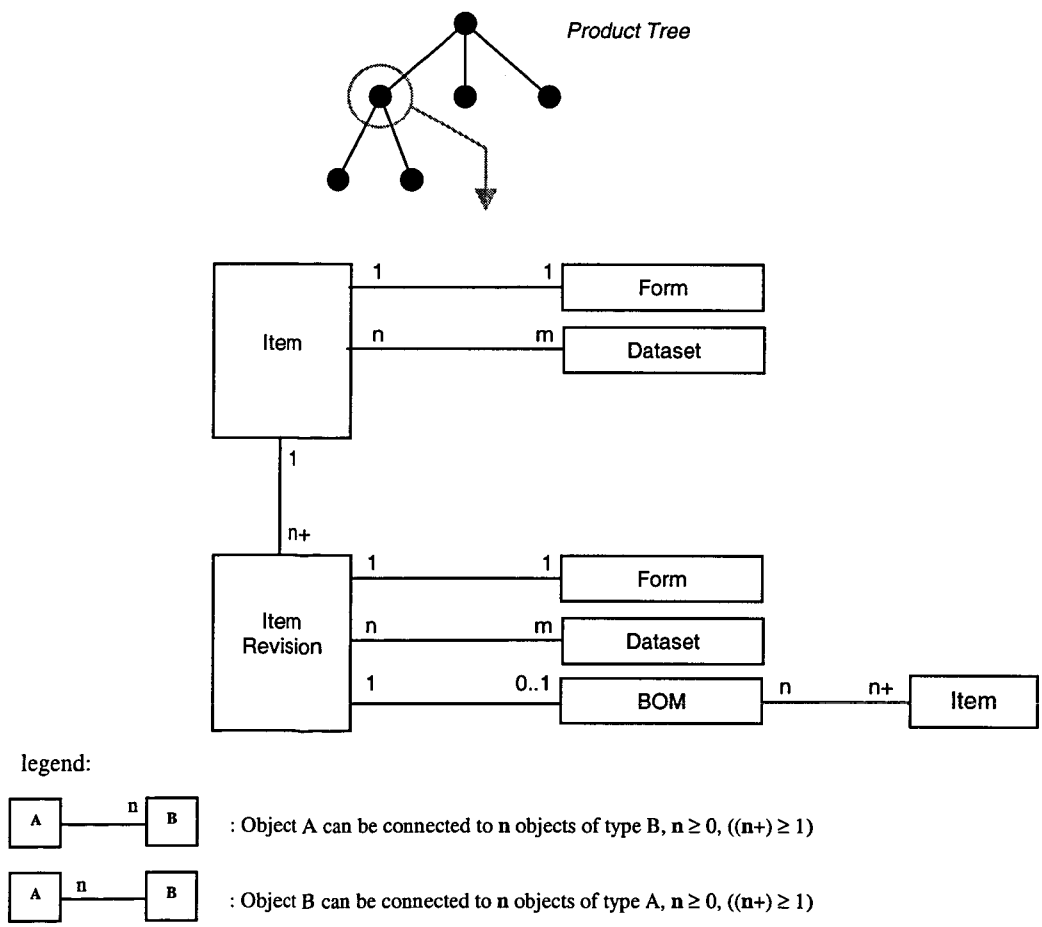


fig. 3.2: Object model product structure

There are some constrains, which are not shown in the figure. An Item-Revision object is always connected to its original Item object from which it is derived. An item-revision can't exist without a link to the original Item and it is not allowed to be connected with another Item object.

Object BOM holds links to items, which represents the components of the item in question. The BOM has to refer to at least one item, which is of course different than the item containing the BOM or from which the item containing the BOM is derived. It should not be possible that a component consists out of parts, which are consisting the component itself again. Looking to the product tree this means that cycles in the tree should be prevented (fig 3.3).

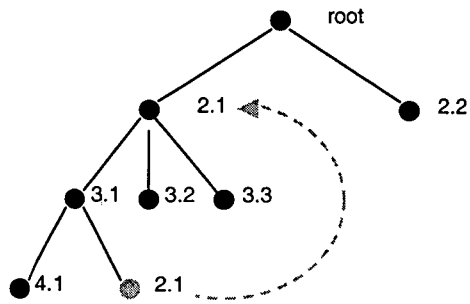


fig.3.3 : unwanted cycles in product tree

BOM constraint:
Let P be the collection of items, $i \in P$ is an item/node in a product tree. Let $B(i)$ be the collection of items in the BOM of i, (children of node i).

For $B(i)$ counts: $B(i) \neq \varnothing \ \wedge \ B(i) \subset \ (P / \{ path(i, root) \})$,where

path(x,y): is the collection nodes (including node x and y) on the path from node x to node y.

3.2.1 Relationship Types

There are several types of links between an item (-revision) and a dataset, also called relationships in IRIS. The relationships in IRIS have a semantic meaning to define why a certain object is connected to another. For example if there is a dataset describing the functionality of a component then it must be connected with the referred item using the relationship *specification*. If the dataset describes for example a fabrication method then the relationship *manifestation* will be used. Each relationship has a set of rules, defining which object can be connected with another using the particular relationship. To prevent misunderstanding, relationships don't have a different technical function, they all link objects to each other. One functionality of having different types of the relationships is to group the linked objects with the same semantic contents. In IRIS it is for example possible to show only the connected objects, which describe the fabrication of a component (objects connected with relationship *manifestation*). The relationships between an Item (-revision) and a dataset have to be set manually by the user. It is possible to choose a default relationship for datasets. The following relationships exist:

Requirement
The *requirement* relation associates data to an Item or Item Revision, such as standards, which must be adhered into a drawing or technical requirements for a part. Rules for this relation are as followed:

- An Item or Item Revision must have *write-access* to add or remove a *requirement* relation to an object.
- A Form, a Dataset with version 0, can have a *requirement* relation to an Item or Item Revision.

Manifestation
The *manifestation* relation associates other related data to an Item or Item Revision. This data may, however, be necessary for information, such as analysis of the competing ideas from which a part was originally conceived. The *manifestation* relation also associates data to the Item Revision that contains data derived from the specification data (such as tool paths). Within Philips, fabrication information is a good example of documentation with a manifestation relation to an item revision. Rules for this relation are as followed:

- An Item or Item Revision does not need to have *write-access* to add or remove a *manifestation* relation to an object. A *manifestation* relation can be added or removed from a released Item or Item Revision.
- A Form, a Dataset with version 0, can have a *manifestation* relation to an Item or Item Revision.

Specification
Specification relations are detailed methods, designs, processes and procedures to satisfy requirements. Within Philips this kind of information is mainly indicated as 'functional information' or '110-documentation'. A specification relationship can only be established with an Item-Revision, not with an Item. The reason behind this is that specifications can change drastically from model to model (-revisions). Rules for this relation are as followed:

- An Item Revision must have *write-access* to add or remove a *specification* relation to an object.
- Within Philips it is defined that a *specification* relationship can only be established with an Item-Revision, not with an Item.
- A Form, a Dataset with version 0, can have a *specification* relation to an Item Revision.

Reference
The *reference* relation associates any data to an Item or Item Revision. Examples for this relationship are field reports, trade articles, customer letters, lab notes, etc. Rules for this relation are as followed:

- An Item or Item Revision does not need to have *write-access* to add or remove a *reference* relation to an object. A *reference* relation can be added or removed from a released Item or Item Revision.
- Any object can have a *reference* relation to an Item or Item Revision.
- A reference object cannot *reference* itself.

3.2.2 Errors in the Relationship Types

One of the standard datasets attached to a product item is 'Survey of Changes'. All changes of the item are described in this document. This dataset is connected to the item with the *reference* relationship. For adding or removing the *reference* relation, the user doesn't need *write*-access for the item in question. Thus, everybody, independently of his/her access-rules can disconnect the Survey of Changes from the item. The dataset itself will not be deleted, but the link from the item to the dataset will be removed. Accessing the file from the item will not be possible, because the item has no longer the address of the dataset. If there are no more links to the dataset, then the

dataset will be automatically deleted. This also holds for objects connected to an item using the *manifestation* relationship.

On the other side, unrelated objects can be attached to an item with the relationship *reference* and *manifestation*. Everybody using IRIS, without having *write*-access to the items can do this. For example, an item-revision is only and always connected to its original item. This is preserved by the constraint between the objects item and item-revision, but the relationship *reference* can cancel the constraint. It is possible to attach an unrelated item-revision to an item by using the relationship *reference*. In practise it can be done by copying an item-revision from one item and paste it to another item (menu options *copy* and *paste*). The column showing the relation type between objects is set hidden (default) in the user interface and is not used to show information. The user has to look now at the identifier of the item-revision to distinguish the real revision from the pasted one.
This also holds for other objects, like an object of the type BOM. With the relationship reference it is possible to attach more BOM objects to an item. Besides, the system has a bug, that it can't distinguish the correct BOM from the referred BOM. This can have consequences for the applications working with the Bill of Material stored in IRIS, like CAD applications. Unauthorised users can change the product tree, without having the right to do this. Of course it has to be assumed that a user will not do this on purpose, but mistakes can happen and wrong information can be stored in the database.

Why the relationship *reference* is introduced? In the former specification of the system, it was probably recorded that the user has to be able to make references between any objects. Instead of 'walking' through the whole directory structures it has to be possible to make references between objects for easy localisation and access. A scenario is imaginable that a designer of an electron gun used in a particular TCA wants a reference to another electron gun in another TCA for comparison. Instead of walking through the whole product tree of the TCA, the designer wants a direct link (a bookmark) from the TCA to the other electron gun. Currently this is possible with the relationship *reference*, but this reference between the TCAs is a personal reference of the designer. It doesn't mean that the reference is related with the product component or adding an extra description to the product. It is a personal reference and probably not useful for other users. The relationship reference satisfies the specification that the user is able to make links between objects, but has many bad side effects. References made by an user for personal reasons should be separated from the product data and not be stored in the items and be visual for other users. The system is missing the functionality of external views for users on the logical data.

Why is relationship *reference* used for attaching objects with related information to a product item? Currently, four relationships (*specification, requirement, manifestation, reference*) are available to attach objects, mostly object dataset, to an item. According to the rules some relationship can be used to attach datasets to items ad others only to item-revisions. The dataset 'Survey of Changes' doesn't satisfy the semantic meaning of the relationships *specification, requirement of manifestation*. That's probably why relationship *reference* is used to attach the dataset (occasionally relation *manifestation* is used).

The logic why relationship *manifestation* doesn't need write access for the item is not clear.

The BOM constraint is not assured by the system. It is possible to make cycles in the product tree.

remark:
Within Display Components all the product documentation has a standard name convention, describing the contents of the document, for example documentation with a 100 code label, describes the functionality of a component (=specification), documentation with a 200 code label describes fabrication information (=manifestation). The relationship types for describing the connection between an Item and the attached documentation is redundant information. The code

label attached to the document already describes the relationship between a document and an item. Having different types of relationships to attach datasets to an item should be reconsidered. Currently the semantic meaning of the relationships is not used and leads only to more misunderstandings. Because of the code label of documents it is possible to have one general relationship to attach datasets to items.

3.3 Suggestions to Improve the Data Model

At the moment the product structure in IRIS is too open, indifferent use can lead to unwanted changes of the product tree. The product tree and the way of working with product data is well defined for the user on paper, but is not implemented as *data model* in IRIS. The defined way of working has to be enforced by the user. The system gives the user too much freedom, it is for the system impossible to check if the product tree is correctly constructed. The integrity of the product structure can't be assured by the PDM system.

The relationship types and constraints have to be correctly configured (implementation of Access Manager[3] does not solve these errors). Introducing *Release & Change procedures* has no effect when a user can cut a released document of an item or add unrelated documents to final items. When objects are distributed to other systems more mistakes and errors can occur.

- Introduce one general relationship type for attaching datasets to an item or item-revision (if the semantic value of the different types is not used).

- Don't use extra 'access rights' for the creating/deleting relations between objects. Use the access rights of the objects.
  For example, when an user wants to add a dataset to an item, than use the access rights of the item. If the user has the right to modify the item, then he/she is allowed to attach the dataset to the item in question.

- Investigate if it is a necessary requirement that a user can make personal links between objects (referring to relationship *reference*). In case it is desirable, then the personal links have to be stored separately from the product structure, for example in the users' profile. (It is perhaps also possible to satisfy this requirement to let the user only make links from his home folder.)

[3] Access Manager is an alternative protection mechanism in IRIS. Access Manager manages access privileges, users will no longer change the protection settings of individual objects in the database. Instead, access to data is controlled by a central mechanism (for an entire database) which acts on the type of object being accessed, the status of the object, and the type of user that accesses the object. Access Manager doesn't take care of the settings of relationships.

4. Problem Area: A Shared Product Structure

A multi-national enterprise, like Display Components, designs and builds its products at multiple facilities that are widely dispersed geographically. It is no longer uncommon for the design tasks on a single product to be dispersed to engineering groups located on different continents. In order for these groups to work effectively on the same project, engineering data needs to be distributed throughout the organisation, to engineering and manufacturing sites, and perhaps to external suppliers. The integrity of the changes at each facility must be maintained all the time. Parts of the product structures are divided among multiple sites, but they have to be connected and viewed as one structure.

In this chapter the problem area of the distribution of product data is described. The first paragraph describes the graduation assignment. In the second paragraph, the distribution of the product structure is explained, followed by a real-life case of product data distribution. In paragraph 4.4 the requirements imposed to the system for distribution of data are given.

4.1. Assignment Description II

The PDM systems IRIS are currently operating independently at eleven sites. There is no integration or communication between the PDM systems. Here lies the gap: the product information is not limited to one site, but needs to be exchanged between multiple sites. From this gap my graduation assignment arose.

There has to be a global accessible PDM system with all its functionality, which covers all the sites and treats them as one global 'site' with a centralised view of all product data.

Assignment:
Investigate, starting from the current situation and the user requirements for distributing product information, the possibilities for distribution and make a conceptual design of a distributed PDM system.

Of the basic functionality of a PDM system (described in chapter 2.2), I focus on two aspects: data vault management and product structure management. I will not concentrate on how PDM is implemented in the organisation, but on the technical aspects of data distribution and how product structure distribution among multiple sites can be realised.

The approach of the assignment will be as follows:

- The product information flow will be investigated and the requirements for product data distribution will be derived.
- The current distribution facilities, which are available for the system IRIS, will be analysed to determine if they are suitable for a distributed product structure.
- In case the current distribution facilities are not suitable, a conceptual design of a PDM system will be made, which provides the desired functionality.

First, the desired product data distribution is investigated and the requirements of the system for a shared product structure are set up (remaining part of chapter 4). Facilities for distribution of product data between the sites are available, but these do not offer the desired functionality. In chapter 5, an analysis is made of the current facility for distributing product data. From this analysis it can be concluded that the system does not satisfy the requirements for a shared product structure. With the use of scenarios it is shown why the distribution mechanism is not

suitable. A new system has to be designed. In IRIS, two kinds of data are stored, objects (meta data) and files (physical data). In the design, the storage of meta-data and the storage of the files are taken separated. Two architecture concepts (central and distributed database) for storing meta-data are given in chapter 6, together with an architecture for storing the files. Furthermore, there will be a brief evaluation of the network requirements for both architectures. In chapter 7, the concept of a distributed database is worked out in detail. Modifications of the data model of the product structure are introduced in order to make possible a one logical database. In the last chapter summarises the recommendations given in this report and gives advise for future development of the current system IRIS.

4.2. Product Information Flow

Creation of a product first starts with the design of the product, after the design phase the development of the manufacturing methods starts (manufacturing phase). In both phases information about the product is created. Information created in the design phase is the input for the manufacturing phase (fig.4.1). Both design and manufactory information has to be collected, so that all the information of a product or component, from design till fabrication, can be found on one place.

The created information, during the design-manufacturing chain, has to be accessible by all the involved groups (if they have authorisation).

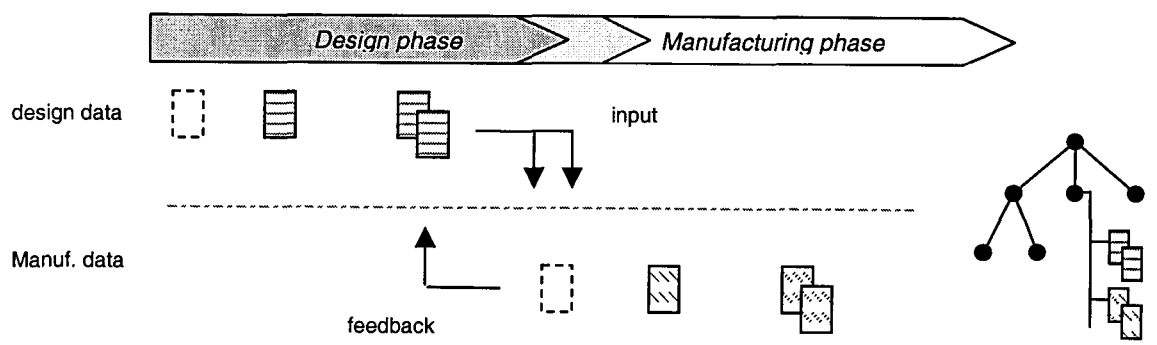


fig.4.1: Product data build up during phases

The information is stored in the nodes of the product tree. The structure of the product tree (the bill of materials) is determined in the design phase. Although the tube coil assembly is basically the same for all tube types, the assembly is not pre-defined. During the development stage, the components are developed separately from each other and related to each other in partial steps. The manufacturing information will be added in the nodes of the (design) tree. This is the current practise, but it is possible that there will be in the near future a manufacturing structure next to the design structure.

Data created at different locations
Design and manufactory of a product will not be done on one location. Currently the design take place in Eindhoven, manufactory however is separated among several locations all over the world. It is even imaginable that the design of a tube coil will be also dispersed among several

locations in the near future. Looking to the product structure this means that there is a division on two levels:

- Assembly level: Components (Items) divided among multiple sites.
- Component level: The content of the component (Datasets) divided among multiple sites.

Assembly level
Components can be developed/manufactured at different locations. Information of the components will be created at and owned by separated sites. Sites have to be able to make relationships (BOMs) between the components (if they have authorisation), also when the ownership of the component is from another site. At the moment every involved site has a part of the structure.

Component level
Information of a component is not limited to one site. More sites in the design-manufacturing chain are involved, which need and create data of a single component. The information of a component is dispersed among several sites.

An illustration of a distributed product structure is given in figure 4.2, where an assembly of a fictional product X is shown. In the figure it is shown that more sites are owned or are responsible for several components of product X (assembly level). For example, site 1 owns the main description of product X (the root) and the sub components A and K and site 4 owns the components L, T and U. On component level, it is illustrated that site 1 (assume a design site) owns the design data of component K. Site 2 (assume the manufacturing site) wants to add manufacturing data of component K in the node of the tree.

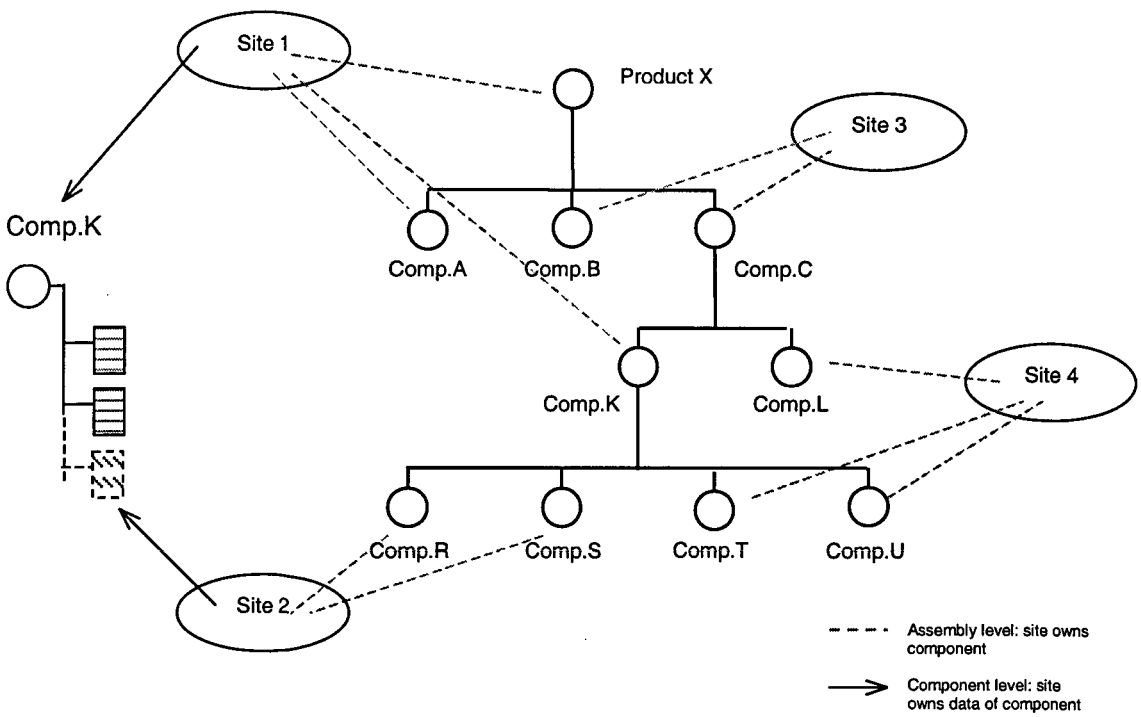


fig.4.2: Distributed product structure

4.3. Case: Data Distribution in Gun Chain

This paragraph describes a real-life case of the distribution of one part of the product structure (the Gun Chain). The Gun Chain is the design and the production chain of the electron gun of a tube coil. The gun is designed in Eindhoven and manufactured on different places in the world. The Gun Chain is a pilot of the TPI way of working in a distributed environment. It concerns the responsibilities and the ownership of the Technical Product Information (TPI) of the electron gun. The concerning parties can be grouped in 4 divisions: PPD (support), IPC (Industrial Product Centre), Gun factory, Suppliers. All of them can be located over the world. The PPD and IPC take care of the development of the gun. The Gun factory takes care about the manufacturing of the gun and Gun Suppliers take care of the gun components manufacturing.

The main issues regarding design and document control in the Gun Chain TPI are the following:

- Separation of the functional data and manufactory data
- Responsibility & Ownership of the data
- Release & Change of design and manufacturing data.
- Security of the data

Separation of the functional data and manufactory data
Division of the total gun data in functional and manufactory data is important to determine who is responsible for the data, who creates what kind of data, who may change data. Separation of the data can be in 3 parts:

- FT: Functional Tube information, information describing the functionality of the tube.
- FG: Functional Gun information, information describing the functionality of the electron gun.
- MP:Manufacturing Product information, information important only for manufacturing of the electron gun and the gun components.

In IRIS the separation of data by function (FT, FG, MP) is taken care of by the relationships connecting a dataset with an item and/or by the document-labels 100-200 (see chapter 3). The Gun component is a sub tree of a whole TCA. The different kind of information concerning the gun is stored on the following levels of the TCA, see figure 4.3.

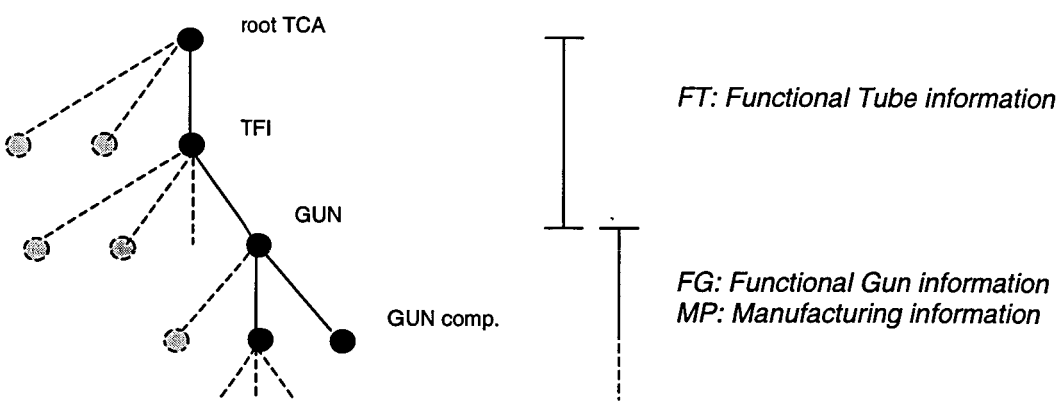


fig.4.3: Storage FT, FG, MP information in product tree

The Functional Tube information is stored in the items on the path from the gun to the root, excluding the GUN item. The GUN item has to be excluded otherwise it is not possible to use the gun in different tube coils (multiple use of items). Functional Gun information is stored in the gun item and the gun component items, together with the manufacturing information of the gun (MP).

Responsibility & Ownership of the data
To be responsible or to be the owner of data are two very closely connected roles. The responsible person and owner are defined as follows:

- *Responsible person:* responsible that the product information is correct. This means: In accordance with the actual produced product and in line with agreements made in the chain and business axis. It is not essential that this function is defined within the IRIS system. This is in fact the function of Change Manager of the IPC.
- *Owner:* Control of the TPI in IRIS. This means: Responsible that the TPI is technically up-to-date and that accepted modifications are introduced in the documentation including the consequences for other products. This is in fact a control function and the name has to be recognisable in IRIS.

Ownership of the data
The ownership of the data is defined during the development and the manufacturing phase. The owner is responsible for keeping the TPI technically up to date and that the accepted modifications are introduced in the documentation including the consequences for other products. Ownership of the data concerning the gun is defined as follows:

Ownership data	Development phase	Manufacturing phase
FT (functional tube)	Project team decides: • First of a kind development : PPD • Second kind of development: IPC	IPC
FG (functional gun)	Project team decides: • First of a kind development : PPD • Second kind of development: Gun factory	Gun factory
MP (Manufacturing data of gun and of gun components)	▪ Gun factory / Supplier in case of strong local development ▪ Otherwise split ownership functions between gun factory (owner) and PPD (technical control)	

Two levels of ownership can be distinguished in IRIS, owner of a dataset (physical data) and owner of an item , item-revision (meta-data). In IRIS every object can have an owner. It still has to be defined how ownership will be divided among the objects. A person can be owner of object Item or item-revision, other persons can be owner of dataset objects attached to the item.

For example: The owner of an item can be seen as the responsible person to check if the information (datasets) attached to the item is technically correct and up-to-date. The owner of a dataset only controls the document, which the dataset represents. Division of ownership within an item concentrates in the division of the datasets. It should be possible to make groups of datasets (development, manufacturing) within the item and appoint an owner to a group. The different data groups (FT, MP) can be easily formed with the relation-functionality. Appointing an (general) owner to a group is not a standard functionality in IRIS.

- It is clear that the ownership of the information is divided among the concerning parties. Ownership includes also that the owner can modify the information stored in IRIS.

Responsibility for the data
Being responsible of the data is not a defined function in IRIS, but it is clear that the responsible person needs access to (all) the information.

Release & Change
In order to work on a consistent way with each other and keeping everybody informed about changes, procedures have to be made. (see appendix II, III)

Security
The security issue is about whom may access (viewing) the information.
Limitations to accessibility are set as follows:
- FT : accessible for everybody
- FG: accessible for everybody
- MP: accessible limited, only for concerned manufacturer

4.4. Requirements for Distributed-IRIS

As seen in the previous paragraphs the main property of the desired PDM system is that the product data is accessible for all the sites divided over the world and that the integrity of the data is maintained during modifications. The desired system has to make it possible to construct a shared product structure. Parts of a product tree can be created at different sites and taken together to assemble one product tree. Ownership of the parts stays in hand of the creators (or transferred to the proper person). Other sites can add data to the product structure and publish this information for other users.

In the following paragraph the basic requirements of the system are described. The requirements can be divided in two parts, user requirements and system requirements. The requirements are independently of the database architecture. Architectures will be discussed in the next chapter.

Local data will refer to data that will not be shared with other sites. *Global data (published data)* is data that is accessible by all sites.

4.4.1 User Requirements

From the previous paragraphs it can be deduced there are two basic functions. The user has to retrieve information and the user has to share it's own information with others. Issues as, separation of functional and manufacturing data, responsibility of the data and ownership of data mentioned in the Gun Chain can be accomplished with a shared product structure accessible by all the sites. In this paragraph the basic operations which are needed for a shared product structure will be specified.

Retrieve information
Accessing information is the primary functionality of the system. To view the data, the user has to locate the meta-data of the wanted product first. The meta-data contains next to the attributes describing the product, the storage address of the file. After locating the meta-data, the user can get the physical data and open it using the tool committed with the data file.

Finding the meta-data can be on two ways:
- Using a find query
- Using the directory mechanism

Find Query mechanism:
The user can locate information in the database using a find query. The options of the find query will be at least the same as on the current local database system.

Directory mechanism:
The functionality the directory mechanism of the global system will be the same as the local systems. The user have to able to 'walk through' the directory structures and product structures. Furthermore, the basic functions (copy, cut, paste, etc) have to be available. Every project will have an own directory where the data of the project can be found.

Publish information
Every project will have a location at the global database. With the directory mechanism a project folder will be prepared for storing the information. *Publishing* in this context will be seen as making the information available for all users.

Publish stand-alone data
Stand-alone data will be viewed as data without any relations to already existing global data. This can be a whole product tree, or only information of a single component. *Stand-alone* doesn't refer to the number of data files, but to the relation with other existing data. There is no relation with the existing data. The user can publish stand-alone data in the project-predefined location. The user locates the project folder and gives the command to publish the information at the selected location.

Add new data to existing component
The user wants to add information to existing data. The product structure stays unchanged, the user adds only new information to a component (fig.4.4).

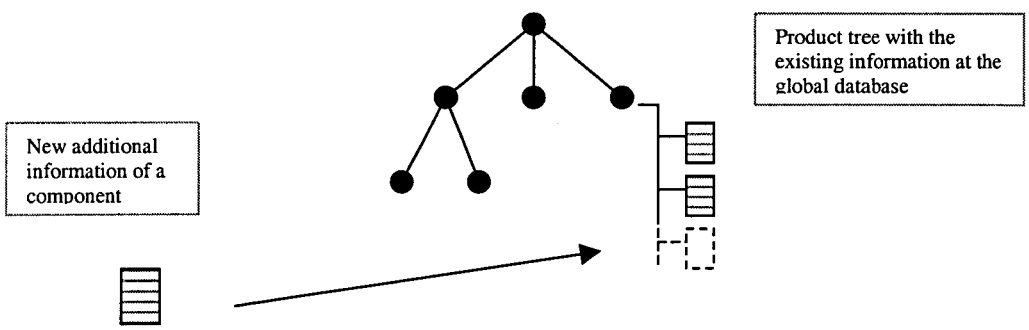


Fig.4.4 : Adding data to existing component

Combining existing data and construct new product structures
A product tree consists of product components (nodes). Development of the components can take place at different sites. These components or sub-trees will be published at the sites and then will be assembled to one product tree. Or a product tree is partially constructed and at a later stadium the remaining part will be added. Several scenarios are imaginable when modifications can be made to the product tree. Bills of Material (BOM) are used to make structures.

Adding a BOM to a leaf node

The user has to be able to make a new product structure (BOM). The new Bill of Material will consist of nodes, which are already published on the global system. The user selects the required components and groups them together in a BOM (fig. 4.5).

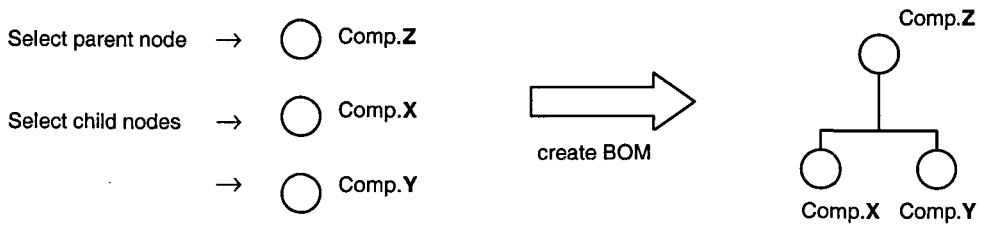


Fig.4.5: Creation BOM

Modifying an existing BOM

Modifying an existing BOM is the same as creating a new BOM, only in this case the new BOM will be a BOM revision of the existing one. Modification to the BOM means creating a new one (new revision), all the revision will be stored in the database. Keeping the old BOM revisions is necessary for accessing other components. In the BOM the links to other component (the children) are stored. It is imaginable that a new BOM doesn't consist of all the children of the old BOM. Thus, the children, which are not part of the new revision can't be accessed anymore through the new BOM. By keeping the revisions, all links to the nodes will be maintained.

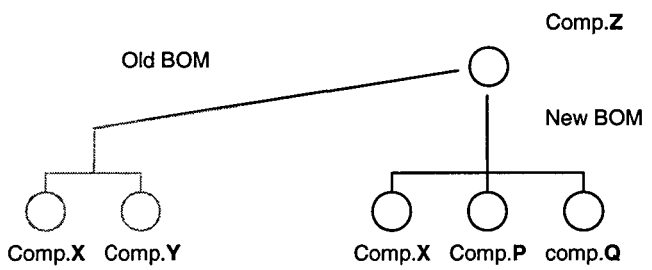


Fig.4.6: New BOM revision

A new Bill of Material is created for comp. Z, consisting of comp. X, P, Q. Comp.Y doesn't take part in the new BOM (fig. 4.6). Accessing comp.Y will be via the old BOM, which is also stored in comp.Z. Creating a new BOM doesn't affect the linked data of the existing structure. The user can always trace back the older version of the structure and access the data, in case there is no link in the new version. If a site connects data with comp.**Y**, adding a new BOM doesn't effect the connections between the existing data.
On the other hand a new revision has definitely effect on the product description, actions on the concerning content of the product have to be taken by the user.

4.4.2 System requirements

The system has to fulfil certain requirements so that the user can operate the system on a comfortable and friendly way.

Preserve the integrity of the data
Preserving the integrity of the data is probably the most important feature of the system. The user has to rely that the information stored in the system is correct and up to date. In the developing stage a product specification is subjected to many modifications. Especially in a current engineering environment it is important that these changes are noticed for all the concerning parties, so that the users can adapt as soon as possible to the new specifications. The system has to assure that when modifications to the product data are made and approved for publishing that all the users are notified of the changes. The system has to assure that all data stays consistent. All (global) data of a product X has to be the same at all sites.

One of the best ways to preserve integrity of the data is to reduce the replication of data to the minimum. All replicas have to be updated when a modification has occurred. The user has to be aware that the data is consistent till that moment of time when the last update took place. The system takes care that update are done as soon as possible (or within an acceptable delay) when necessary. In case the network connection between the local site and the global system fails then the user has to be made aware that all the global data, which is still accessible, is consistent till the moment of network failure. When the network failure is fixed, the system has to update the modification of the data, which were made during the network failure.

Uniqueness ID's of objects
Object Item represents a product or product part. The system has to assure that the ID of an item object is unique at every site, this also counts for Item-Revisions.
This uniqueness of item ID's can be narrowed to the uniqueness of the item ID's with the 12NC format[4].

Local and global data:
The user wants to add local information to global information, with the preservation of the relations between the data. The local data has to be in relation with the global data. Local and global data has to be accessible by the same application.

Performance
Performance goals have not been defined yet, but it is reasonable to say that the system provides the same performance as the current system.

High Availability
Due to the geographicly dispersed locations of the sites the system has to be 24 hours operational to cover the time zones. Downtime of the system through maintenance or failure has to be avoided or at least kept as short as possible.

Scalability
The system has to be able to grow in processing power and data storage capability.

[4] 12NC format : A 12 numeric code, uniquely identifying product/components. The 12NC is extracted from the system PRIME, which register the product / component with the unique number.

5. Current Distribution Facilities for Product Data

At the moment there are two ways of distribution; sending the data manually to the sites outside the scope of the system (which is currently the case) and using the ODS facility (Object Directory Services). In this chapter, the distribution facilities for product data, offered by the system IRIS, are described and the shortcomings of the facilities are explained.

5.1. Manual Distribution

Currently data is distributed manually. At the PPD site in Eindhoven a person responsible for the distribution keeps track of all information, which is distributed to other sites. This distribution can be seen as sending a copy of the data from one site to another (outside the scope of the system). There is no synchronisation by the system between the original data (*master object*) and the replicated data. A new version of the data means sending manually a new copy of the modified original to the other site. There is no shared product structure. Parts and copies of the structure are separated among sites and they are not related to each other. Distribution goes outside the scope of the system.

With the use of the *import* and *export* commands in IRIS, it is possible to send manually data from one site to another.

5.2. Automatic Distribution: ODS (Object Directory Service)

A controlled way of distribution of data is supported by the ODS (Object Directory Service). The ODS is based on data replication, copies of the original data are sent to various sites and the copies are updated when changes to the original data are made. Not all published information is useful for all sites. The published data will be placed in a global directory, where each site can select the information it needs with the use of a find query. Once the site has selected the requested information, updates of the original object are automatically sent to this site. The ODS can be best seen as a global directory, where data is published for read-only purposes. It is a 'one-way' distribution; a site creates data and sends read-only copies to other sites.

The ODS has the following 'rules' concerning the distribution of data:

- Only the *master object* can be replicated. A *replica* can't be copied again. When an object is initially created and saved in a database, then that instance is considered the master object until ownership of the object is transferred to another site. The other site will have now the master object.

- Only the master object can be modified. All replicas of the master object are read-only. This ensures that the master object is always the latest version.

- After a master object has been replicated, it can't be deleted until all replicas have been deleted. This ensures network-wide referential integrity.

In order to share information across multiple sites, each site can publish objects to a global directory. The ODS itself is a database, containing records of all objects which are made available. It doesn't contain the objects, but it contains the information about the location of the objects. A site is connected to the ODS, where it can make notifications of the information which it wants to share with other sites. The site can also look for information made available by other

sites on the ODS. The sites are connected with each other for transferring a copy of the master data. The ODS itself is a separate database system, more sites can be connected to one ODS. (fig.5.1: ODS architecture).

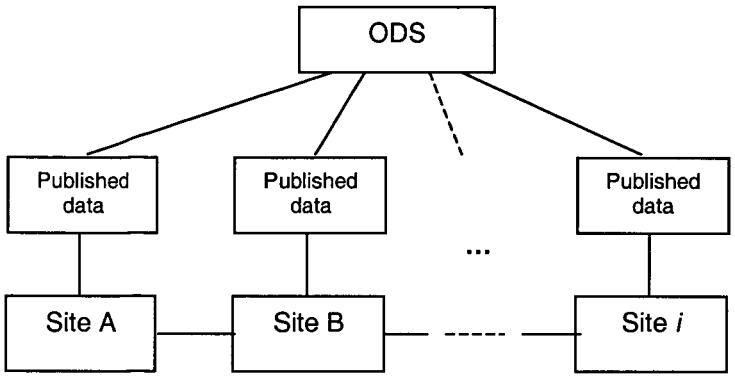


Fig.5.1: ODS architecture

There are two main functionalities, which the ODS offers:

- Making information available for other sites (*publish information*)
- Retrieving information from other sites

Publish information
A site can make local information global by using the command *publish*. When data is published, a record, containing the minimum attributes for locating the object, is stored in a global directory (ODS). The ODS is an union of all publication records of all sites. Once the publication record is stored at the ODS, other sites can see this record and retrieve a copy of the original data. Only the master object can be published.

Retrieve information
For retrieving information a site first has to locate the publication record at the ODS. Information can be found by a find query on the attributes of the record. Once the record is located, the site can retrieve the wanted data. A connection with the owning site of the data is made and a copy of the data is transferred to the requesting site. The ODS is like a telephone book, usable for looking up the address of the requested data. After looking up, the site makes a direct connection with the owning site of the requested data and retrieves a copy of the master. This replica will be stored in the requesting database. The owning site keeps track which sites have a replica of the master. The sites will have a 'subscription' of the object and receive, every time the master is modified, a read-only update.

What kind of information can be shared among sites?
Basically all objects can be published, but it is mainly used for distributing Items with their contents. Copies of the master item with the attached objects can be distributed among multiple sites. The Bill of Material of an item can be published, in case the site owns the master objects of the children. The children have to be made global too.

The ODS is designed for one way distribution of information. The objects can only be published once and the published objects are read-only. Other sites cannot modify the published objects. The restrictions of this one way distribution will be shown in the next paragraph.

5.3. Limitations of ODS

The ODS has certain limitations, which makes the use of the ODS for a shared product structure not suitable. In this paragraph the limitations of the ODS will be shown by scenarios. The scenarios are not completely tested with the system, but are derived with the rules of the ODS.

The ODS is a 'one way' distribution tool, it only distributes read-only copies of a master object to other sites. The other sites can have a so-called subscription to the object and receive new read-only updates of the object when the object is changed. The product structure has two *shared* levels, *component level* and *assembly level.*

Scenario Shared Component:

Assuming 3 sites (2 design sites *D1* and *D2*, 1 manufacturing site *M1*) working on one component *Item X. D1* starts the design of component X and creates *Item X* with the first documentation. *D2* wants to add documentation to *Item X.* Site *M1* wants to access all the available information of comp. *X.* (fig.5.2)

Using the ODS, scenario 1 can be realised as follows:

- D1 publish *Item X* with all the content (from the view of site D1 this is only *dataset 1*) on the ODS.
- D2 takes *Item X* of the ODS, including the content.
- D2 adds *dataset 2* and *dataset 3* (dependable of what kind of relationship is used) and wants to share this with the other sites. In order to do so, the datasets have to be published separately.
- M1 wants to read all the information of *X.* In order to do this it has to take *Item X, dataset 2* and *dataset 3* from the ODS and makes the relationships between the *datasets 2, 3* and item *X* manually.

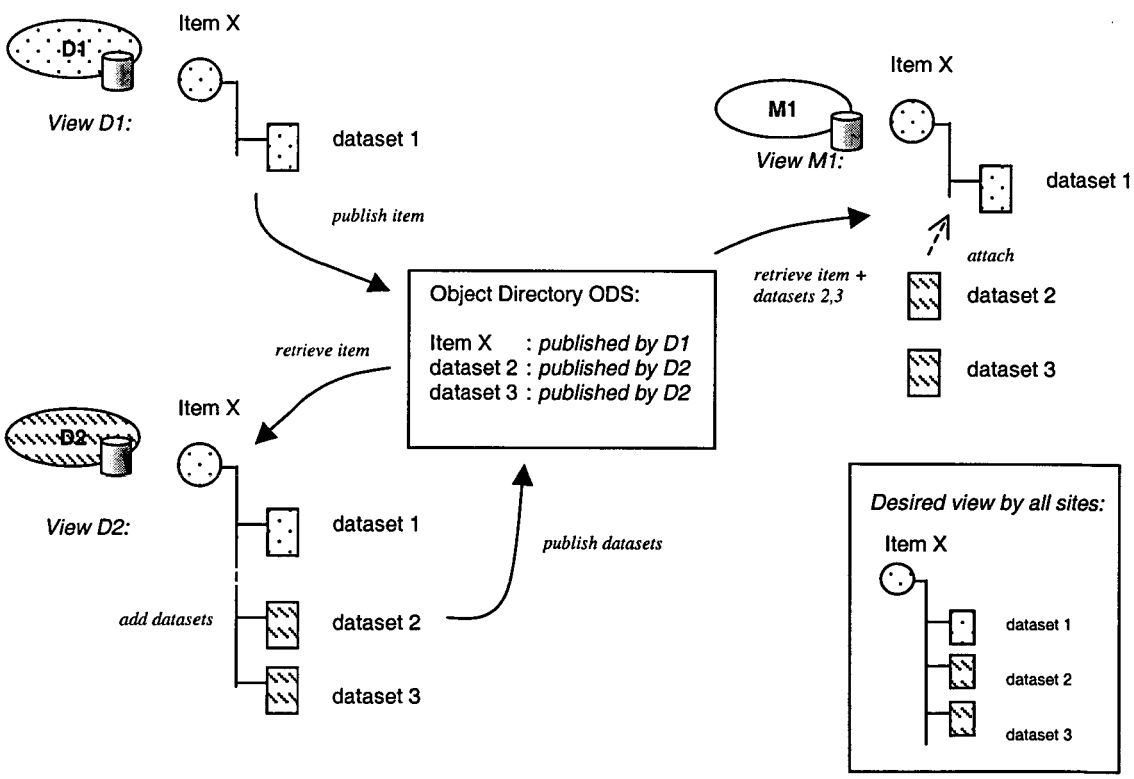


fig.5.2: Scenario Shared Component

- In case site D1 wants all information, then it has to retrieve dataset 2 and 3 from the ODS and attach it manually to Item X.

The scenario is a simple illustration how a shared view can be built up with the use of the ODS. The problem is that site *M1* needs to know that *dataset 2* and *dataset 3* have to be connected with *item X*. This means that there has to be some kind of document describing which datasets belongs to which Item.

In this scenario the item-revisions are neglected, if those are introduced then things become much more complicated. Suppose site D1 creates a new item-revision, the replicas of item *X* will be updated with the revision. All the other sites have to be notified that they have to connect the *datasets 2,3* to the new revision. It becomes even more complicated if site *D2* wants to make an item-revison, then the single revision has to published on the ODS. All the other sites have to collect the correct information again to attach it to this new revision. This new revision is part of the Item owned by site *D1*. Site *D1* has to add this revision to its own Item and then add all datasets to it. Sending replicas of item *X* to other sites is not possible anymore. The item *X* contains a new revision owned by *D2*. *D1* can't send replicas of replicas. To solve this site *D2* has to transfer ownership of the revision to site *D1*.

It is clear, that just for one component, a lot of communication and strict documentation is necessary to have at all sites the same view of the product structure.

Scenario Shared Assembly:

2 design sites *D1* and *D2*, working on one product *Y*. The product consists of 3 parts K, L, M. Site D1 owns the root of the product and component K and L (Items Y, K, L). Site D2 designs component M. *D1* starts the design of product Y and creates *Item Y, K, L* and relate the items with a BOM. D2 designs component M and want to relate item M to the rest of the product structure (fig.5.3).

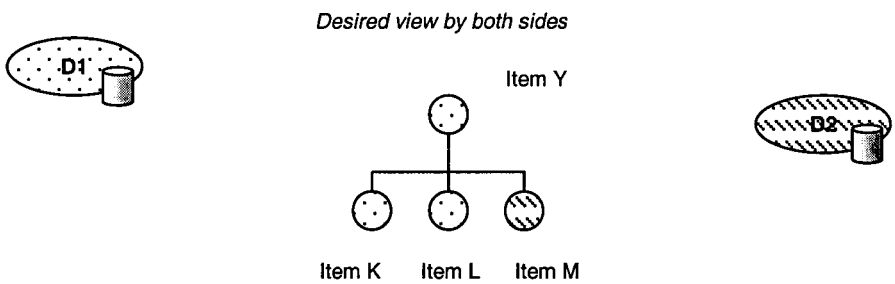


fig.5.3: Scenario Shared Assembly

This scenario is not easy with the use of the ODS and perhaps it is even impossible to give all the sites the same view of the structure, while the ownership of the separate components remains at the sites. The explanation will be briefly, just to pinpoint the problem. The problem is that it is not possible to publish relations to other objects published on the ODS. It is not possible to publish a BOM object with links to an item of another site, for example to publish Item Y with a bill of material containing the items K, L, M.

Suppose *D2* publishes item *M* and site *D1* retrieves a replica of item *M*. This replica will be stored in the database of *D1*. *D1* can probably make a new BOM of item *Y*, containing a link to the replica of item *M* stored in the database of *D1*. Here is the problem, when site *D1* wants to publish Item *Y* with the new BOM, the system will probably start to complain that it is not possible

to publish a replica of item *M*. The link in the BOM is to the replica and not to the object on the ODS. Ways to get around this problem are complicated and not easy to do, therefore they will not be mentioned.

What is possible with the ODS ?

The ODS is a tool for distributing stand-alone objects (see user requirements). A site can distribute replicas of its own information to other sites. The other sites have the possibility to combine the replicas with their own local information, but when they want to distribute this local information in combination with the replicas, the problems occur.

The ODS is not suitable for shared product structure. The ODS can be used for 'one way' distribution of data between two sites. Site A works on a product, distributes copies to side B, which is next in the chain. When site A is finished, it transfers the ownership to site B, which can distribute now the data with its own data to other sites again. It is like a relay, where the site with the baton owns all data and has the right to distribute the data. If another site wants to distribute its data then the baton has to be first transferred to this site. This is the only way to assure that the data of a product stay connected and kept together in one structure.

Furthermore, the acts to distribute objects with the ODS are complicated and complex with the user interface. The user has to fill in many parameters and mistakes can easily happen. When the ODS will be taken in use this could be solved by designing a new user interface for the ODS with a default parameter setting and with a simple command to publish objects.

6. Architectures for a Globally Accessible PDM System

In chapter 4 conclusions have been made that the product information stored in the PDM system has to be accessible by all the involved sites. In the previous chapter it is shown that the current PDM system IRIS doesn't provide this requirement. In this chapter, two architectures are introduced, which provide the global accessibility of the product data. The PDM system IRIS manages two kinds of data, *meta-data* (the data objects) and *physical data* (the files). The meta data holds the link to the physical data. The architecture concepts proposed in the first paragraphs of this chapter are related to the storage of meta-data. Storage of the files is explained in paragraph 6.5. In paragraph 6.6, a brief evaluation of the network requirements is given for both architectures.

There are two kinds of database architecture concepts for making the data global accessible, a distributed database system and a central database system. The first concept is to maintain the current database systems on their location and connect them to form *one logical database.* This means altering the current system so that there will be communication between the local database systems. Next to this concept, the question arose if it is necessary to maintain all local database systems. Is it possible to construct a database system on one location reachable by all other sites? Both concepts will be discussed in this chapter. The two architecture concepts will be compared and the advantages and disadvantages of each concept will be given. An impression of each concept is given in figure 6.1.

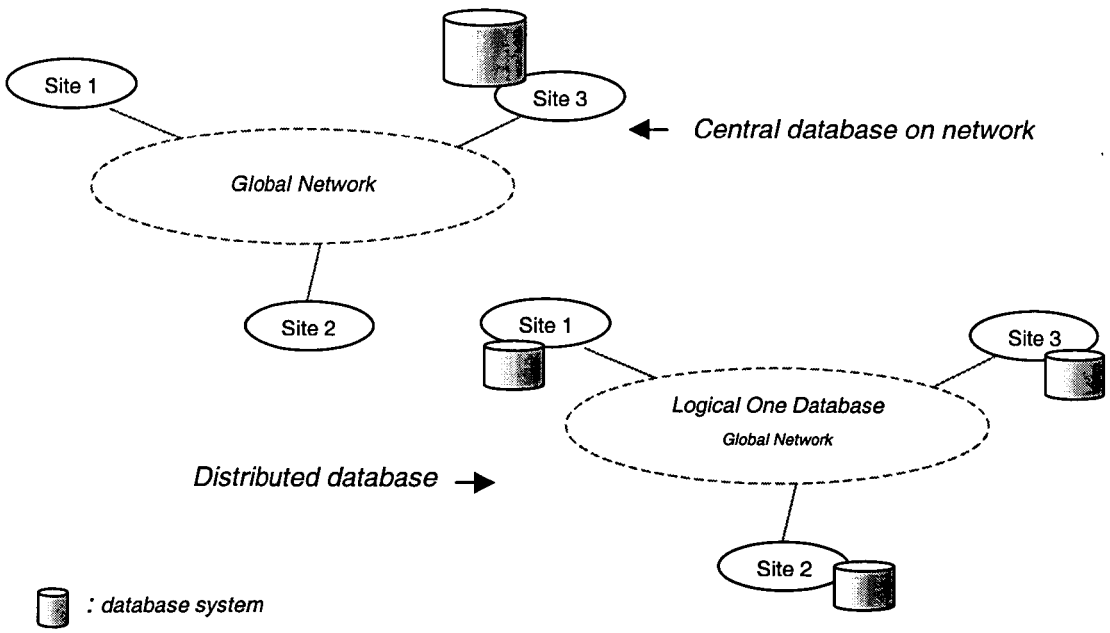


fig.6.1: Architecture concepts

The architecture of a database is greatly influenced by the underlying computer system on which the database system runs. Aspects of computer architecture, such as networking, parallelism and distribution are reflected in the architecture of the database system. Networking of computers allows some tasks to be executed on a server system and some other tasks to be executed on client systems. Parallel processing allows database system activities to speed up, allowing faster response to transactions, as well as more transactions per second. Distributed data across sites or departments in an organisation allows those data to reside where they are generated or most needed, but still be accessible from other sites.

6.1. Aspects Determining Architecture

Determining an architecture is not straightforward, many aspects have to be taken into consideration. In this paragraph several aspects will be mentioned.

physical data – meta data

The PDM system IRIS involves two kinds of data: *physical data* and *meta-data*. The PDM system manages the files (physical data) generated by the applications (Word, CAD, etc.), which are integrated with the PDM system. The meta-data is the information about these files. In the architecture these two kinds of data are taken separated[5].

The size of the physical files managed by IRIS can vary from dozens of Kb to several Mb[6]. Just transporting these files over the network definitely has a big impact on the network performance. It is advisable to store the files close to where they are used the most, which is close to the owner on the local network. This holds for both architecture concepts. (see paragraph 6.5)

local data - global data

Not all information stored in IRIS has to be shared between all the sites. Distribution of information is from design sites to particular manufacturing sites, which needs the design information and visa versa. The data produced by a manufacturing site is in most cases local and has no value for other manufacturing sites. The manufacturing data has to be exchangeable if needed, but requests from other manufacturing sites will not be frequent. The data will only be shared between the sites in the development & manufacturing chain (see chapter 4).

Next to the distribution of official TPI, there is also data stored in the system, which will not qualify as official TPI. IRIS is, next to the PDM functionality, also a design tool, a working environment for the user. Before the user (especially users of CAD systems) delivers official TPI, many 'test' data have preceded (and have been stored in the system) before official TPI originated. This 'test' data has probably no value for other users and is mainly used only by the owner. This data will definitely not be shared outside the site, at least only between the members of the project group. This particularly holds for the physical data. Large CAD files are 'test' files and not are classified as official TPI. These files will never be shared with other users, but have probably important value for the owner.

availability

Due to the geographic location of the sites in different time zones, the system has to be operational almost 24 hours a day. This means a high availability system. During operation time it also has to be possible to perform some kind of maintenance to the system, like making back-ups.

performance

Performance goals of the system have not been defined yet, but to give the user "an interactive feeling" there has to be a "fast" response time of the system to queries of the user.
The network will be the limiting factor in performance, especially when the physical files are frequently transferred over the network. Details about the network requirements will be discussed in paragraph 6.6.

[5] In the scope of this document the term *data* refers to *meta data* and *physical data* refers to the files managed by IRIS.
[6] Size of the CAD files varied between 0.5 MB to 3.0 MB (small evaluation files stored in IRIS).

scalability

The system IRIS is not fully in use at the moment. The system is still in the introduction phase. The number of users is still increasing and the size of the data stored in IRIS will only go up. The system has to be able to grow in size and processing capability. Scalability of the system is therefore a pre.

6.2. Central Database System: Meta Data Stored on One Location

In this paragraph an introduction of central database architectures is given in compliance with the aspects mentioned in the previous paragraph.

A central database system in the scope of this document is a system, which is physical located on one place. Physical on one location means that the database application and storage of the data is on one site. All the remote sites will login on the database site. The main advantage, especially in the view of the PPD, to have a central database system is that the management and the maintenance of the system happens on one location. The bottleneck of this concept will be the network bandwidth between clients on the remote sites and the database site. All the requests of the clients at the remote sites are routed over the global network to the database site.

One of the primary requirements of the system is the high availability. The system has to stay 24 hours a day in the air. Normal system operations such as database backup and partial computer system failures should not interrupt database use. A parallel system is a protection against failure of a database machine, whereas application- and database servers may have backup hardware available so that the software can run on a new machine if the server fails. Beside the high availability of a parallel system it also increases the performance. Although increasing the processing capability has no effect if the network can't handle all the clients' requests.

An overview of parallel systems is given in paragraph 6.2.2. First the current architecture model, client-server model will be discussed.

6.2.1 Current Architecture: Client - Server Model

Centralised systems today act as server systems that satisfy the requests generated by client systems. Client-server is the concept where tasks are divided between clients and servers. The tasks are performed by those computers, which are most suitable. The concept is based on a dialog between the client and the server. The client machine provides front-end application software for accessing the data on the server. The client initiates transactions; the server processes the transactions. There is a structured query language (SQL) that can be used to access data stored on the server side. The network enables remote data access through client-server and server-to-server communication. Taking the centrally located database concept with a lot of remote clients, it is important that the communication between the client and the server is as 'thin' as possible. The size of the communication packets has to be small to keep the network load to a minimum.

Client
The client portion is the *front-end* database application and interacts with a user through an inter face. The client portion has no data access responsibilities; it concentrates on requesting, processing and presenting data managed by the server portion.

Server
The server portion runs the database application software and handles the functions required for concurrent, shared data access. It is often referred to as the *back-end*. The server portion receives and processes queries originating from client applications.

The hardware architecture of an IRIS installation exists of a client-server cluster (fig.6.2), which contains the following:

- client: PC or workstation containing the user interface
- Application server: execution of the application IRIS and the Oracle application.
- Database server: execution of the database and the Oracle installation
- File server: storage of the physical data, the documents, which are managed by IRIS.

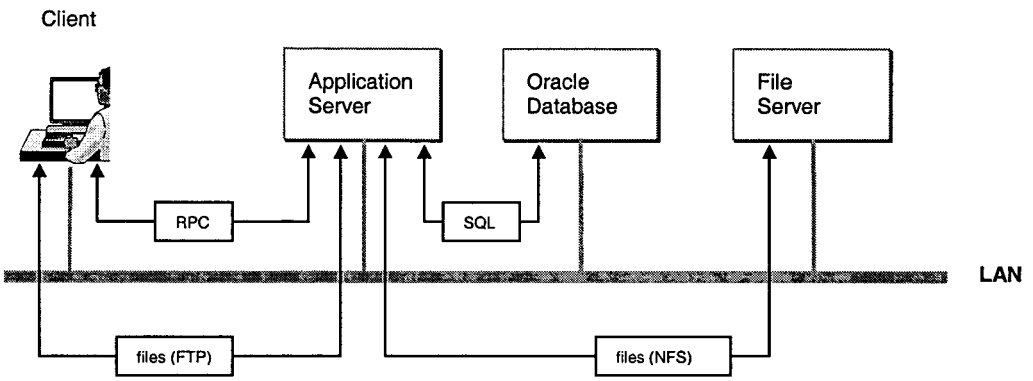


fig.6.2: IRIS hardware architecture

remark:
It looks like that transport of files from the client to file-server will pass by the application server and that there is no direct connection between the client and the file server. The application server is the accessing point of the client to the meta-data and the physical data. This is an assumption; I was not able to trace the precise communication between client and servers.

6.2.2 Parallel Database Systems

Parallel processing in brief
Parallel processing exploits multiprocessor computers to run application programs by using several processors co-operatively. Often multiple processors reside on a single machine. Parallel systems improve processing and input/output speeds by using multiple CPUs and disks in parallel. In parallel processing, many operations are performed simultaneously; as opposed to in serial processing, in which the computational steps are performed sequentially. Parallel database systems exploit the parallelism in data management in order to deliver high performance and high availability at a much lower price than equivalent mainframe computers. Some tasks can be effectively divided and thus are good candidates for parallel processing. Other tasks, however, do not lend themselves to this approach. For example, in a bank with only one teller, all customers must form a single queue to be served. With two tellers, the task can be effectively split so those customers can form two queues and are served twice as fast. This is a small example in which parallel processing is an effective solution. By contrast, if the bank manager must approve all loan requests, parallel processing will not necessarily speed up the flow of loans. No matter how many

tellers are available to process loans, all the requests must form a single queue for bank manager approval. No amount of parallel processing can overcome this built-in bottleneck of the system.

Effective implementation of parallel processing involves two challenges:
- structuring tasks so that certain tasks can be executed at the same time (in parallel)
- preserving the sequencing of tasks which must be executed serially

A parallel processing system has the following characteristics:
- Each processor in a system can perform tasks concurrently.
- Tasks may need to be synchronised.
- Processors usually share resources, such as data, disks and other devices.

A variety of hardware architectures allow multiple computers to share access to data, software, or peripheral devices. A parallel database is designed to take advantage of such architectures by running multiple instances which "share" a single physical database. In appropriate applications, a parallel server can allow access to a single database by users on multiple machines, with increased performance. A parallel server processes transactions in parallel by servicing a stream of transactions using multiple CPUs on different computers, where each CPU processes an entire transaction. In addition to balancing the workload among CPUs, the parallel database provides high-availability. Failure of one of the multiple computers doesn't bring down the whole system. The remaining computers can continue to provide data access to the users.

Without going in details of parallel processing and parallel computers, a few architecture configurations will be given based on Oracle database technology.

The hot standby database

The hot standby database can be best described as a database that is in a state of infinite media recovery. Parallel processing is not directly applied in this architecture, but this is a well-used concept to provide high-availability. The strategy is to create a backup of the database on a second machine (*mirror*) and to ship the logs with the modifications of the data (*redo logs*) to the backup machine, where they are applied to the backup machine (figure 9.3). The backup machine will be a copy of the primary database. On the event that the primary database fails, the hot standby database will take over and will be accessible for the users. The primary system will become, after fixing the failure, the backup system. The architecture only answers to the high availability requirement.

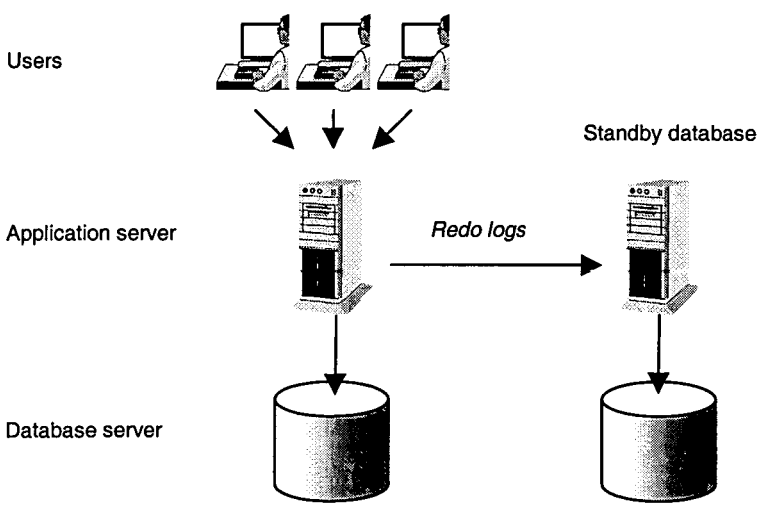


fig. 6.3: Hot standby database architecture

The advantage of the standby database is that it is very easy to set up and operate. Scripting the job to copy archived redo logs to the backup machine is trivial and switching to the standby system in case of a failure can be in short time. The disadvantage of the backup system is that the system has to be of the exact same capability of the primary system. The system has to be held as reserve for event of failure, doing nothing else then applying redo logs. The processing capability of the second system will not be used and therefore it is a relative expensive solution for high availability.
To compensate the not-used capacity it might be possible to open the hot standby database in read-only mode, on the way that read-only queries can be divided over both systems (Oracle version 8 offers this kind of functionality).

The parallel server architecture

The concept of the parallel server architecture is that two or more sessions generated on different machines, open and manipulate data on one database (see figure 6.4). This architecture offers high-availability to the processing machines, because if one of the machines fails processing can continue on the other machines. The performance will be lesser during failure, but still the system is operational. Beside the high-availability, this architecture also offers scalability. More processing power can be added.

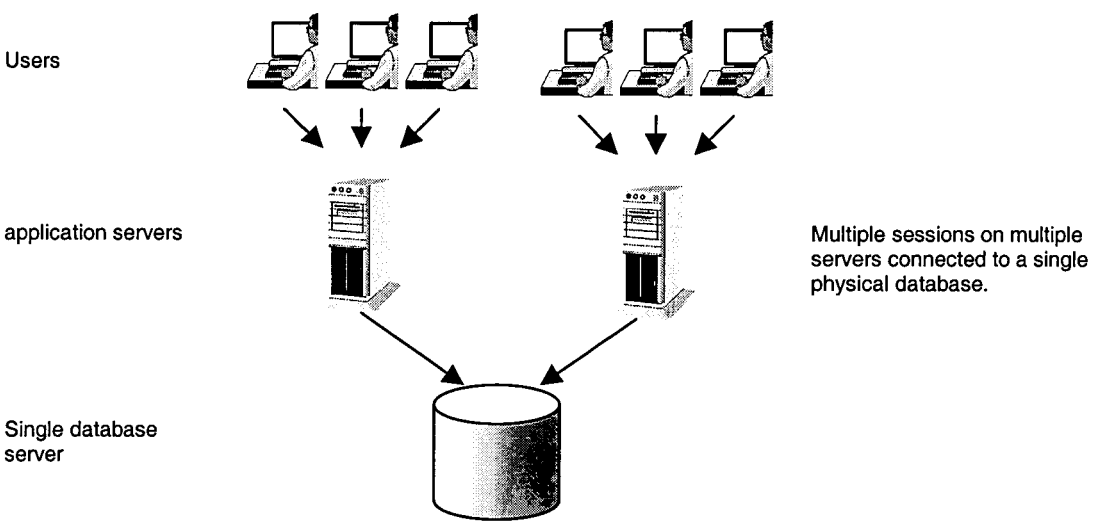


fig. 6.4: Parallel server architecture

The theory of parallel systems sounds fantastic: it offers high availability by multiple machines and it can scale since machines can be added. Alas, not all applications are suitable for parallel processing. The downside of this architecture is becomes clear when two or more sessions on different machines are accessing the same object. A user accesses an object through one of the application servers, while another user can be modified the object through the other application server. When this happens, the data has to be written from the modifying server back to the database, from which the requesting server reads it again. In case many objects are accessed simultaneously by multiple servers, synchronising the objects can be very expensive and has definitely an effect on the performance. The application has to be designed in such a way that it minimises the synchronisation of the objects.

In short, a parallel system can be an ideal solution for both high availability and scalability, but the benefits are not automatic. It has to be researched if IRIS is suitable for parallel processing.

6.3. Distributed Database System: Meta Data Stored on Multiple Locations

Distributed processing and distributed databases are not the same thing, although they have similarities. In a distributed processing system, processing data (searching for information, storing results) is distributed. In a distributed database system, the data is distributed on several computers. A distributed database system appears to a user as a single database but is, in fact, a set of two or more databases, which are joined together forming *one logical database.* The databases don't have to be located on one site. They can be placed geographic apart from each other. The computers in a distributed system communicate with each other through the existing network. Figure 6.5 illustrate the concept of a distributed database system.

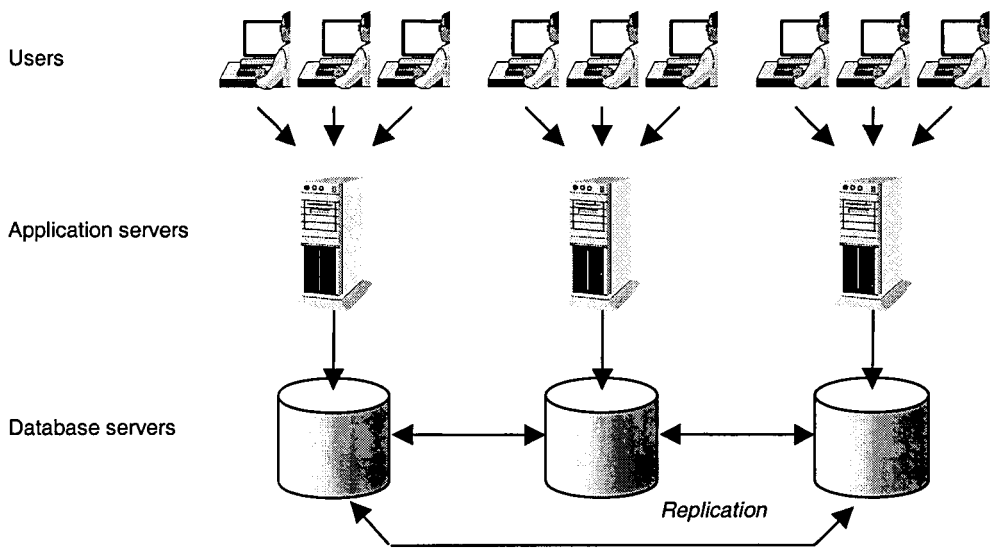


fig. 6.5: Distributed database architecture

Data Replication
One of the ways to accomplish one logical database is *data replication.* Representatives of global data objects will be sent to all the databases. Data replication means that any given data object can have several stored representatives at several different sites. In case the object is modified, then there must be a mechanism for insuring that all representatives reflect the changes.

Local autonomy
In a centralised system, the database administrator of the central site controls the database. In a distributed system, there is a global administrator responsible for the entire system. A part of these responsibilities are delegated to the local database administrator for each site. The sites maintain control over their data and determine their own security settings. Data can be shared without sharing accounts and passwords. Administration tasks can be delegated to the sites, like making user accounts and updating the accounts. Each site can have its own administrator and its own sets of accounts and private data can be kept local.
The possibility of local autonomy is often considered as a major advantage of distributed systems. If this local autonomy is not wanted, the administration jobs can be remotely done from a central place.

Availability
One of the major advantages of a distributed database system is the autonomy of the sites. The database on the site is fully functional regardless of whether it is able to contact its companions. If one of the sites fails, the remaining sites can continue to operate. In particular if data is replicated in several sites, a request for a certain object may be found on another site. Thus, failure of a site doesn't necessarily imply the shutdown of the system. The failure of one site must be detected by the system and appropriate actions may be needed to recover from the failure.

The primary disadvantage of distributed database systems is the added complexity required to ensure the proper co-ordination among the sites. It is more difficult to implement a distributed system, thus it is more costly.

In chapter 7 it is shown how to adjust the data model to accomplish one logical database.

6.4. Comparison: Central Database – Distributed Database

Which architecture for storing the meta-data, central or distributed, is now the best for a shared product structure? Taking the aspects mentioned in paragraph 6.1 and the brief explanation of the both concepts in the previous paragraphs, a comparison between a central database and distributed database architecture can be made.

Local data - Global data
With a central database system there is no real separation between local data and global data. All the data is stored in one database, the system treats local and global data as the same. With a distributed system a distinction between local and global data can be made. Global data will be distributed to other sites; local data doesn't cross the boundaries of the site. The data is stored close where it is used the most, which results in higher performance.

Performance
The performance of a distributed system will be higher than a central system. The data stored in the system is partitioned among multiple databases. The processing power is dedicated to a smaller amount of data. Although the processing power will not be the limiting factor, in both architectures the processing power can be scalable. The main difference will be the network load, with a distributed system 'all' requests are handled on the local network, with a central system all requests will be over the global network. The global network is in general 'slower' than local networks. It can be assumed that the costs of communication of synchronisation with a distributed system are lower than all the remote requests of the users. In paragraph 6.6 more attention will be given to the network requirements.

Availability
Both architectures can be configured to provide high availability.

Scalability
A distributed system is very scalable; it is very easy to add a new database system. With the centralised concept, using a parallel architecture can provide scalability as well.

Complexity
It is clear that a distributed system is more complex than a central system. Higher complexity implies higher costs. Furthermore it is more difficult to configure the database systems and adjust them to each other in order to establish one logical database. Because of the higher complexity, a distributed system will cost more than a central system.

Local autonomy
A local database administrator is perhaps not so desirable for organisational reasons (more personnel, more communication and agreements), but some administration tasks, like making and updating accounts, can be better performed locally. With a distributed system, more local autonomy can be provided than with a central system. It is possible with a central system to let administrative tasks be performed by local administrators by giving them remote access to the central database.

Management and maintenance
Management and maintenance of a central system will be easier, because everything situated on one location. On the other hand, tools are available (or can be designed) for a distributed system to manage and monitor the system also from one location. A distributed system is more tuneable when it is correctly installed. Because of the local autonomy, it is possible to tune the local systems to the local user behaviour. Not every site will have the same behaviour, for example: there will be sites, which read more the data than create it. The physical data is stored locally, so there are always some local maintenance tasks.

Besides, the system will be used 24 hours, so during that time there always has to be a helpdesk available for the users.

Combination architectures
A combination of the architectures is also possible. In a distributed architecture it doesn't mean that the 11 current database systems have to stay alive. It is also possible to take groups of sites (a group per continent) and install for each of those groups one 'larger' database (for example a parallel system) and then connect the large database systems with each other to form one logical database.

Taking all the mentioned aspects the following table can be made:

Aspects	Central system	Distributed system
performance	depending on network	good
availability	high possible	high
scalability	possible in processing power and capacity	In processing power and capacity
complexity	normal	high
management/maintenance	normal	on local sites
costs	normal	higher

6.5. Distributed File Storage: Files Stored on Multiple Locations

In this paragraph the concept of a distributed storage of files is explained. The allocation and the distribution of the files are described and the relation with the meta-data is shown.

The size of the files managed by IRIS can vary from dozens of Kb to several Mb[7]. Just transporting these files over the network definitely has a big impact on the network performance. Current network bandwidth from Eindhoven to the other sites has an average of 1 Mbps, which is not enough to store all files from all the sites at one location (Eindhoven). Therefore, the architecture for storing the physical data will be distributed among the sites. The files will be stored close to the user, so that the user has a fast access to the files.

- *Independent of which architecture concept for meta-data (central or distributed) will be chosen, file storage has to be distributed among the sites.*

An object of the type Dataset represents a file. The storage of the object and the storage of file can be taken separated. The object of the type Dataset contains the address of the storage place of the file. When an user opens (read) a file, the dataset representing the file has to be selected, so that the storage address of the file will be known. A copy of the master file stored at the file-server will be shipped to the users' local environment (the client). The file will be temporary stored on the client. In case the user changes the file (write) and saves these modifications, the changed file will be shipped from the client environment to the file-server, where it will be stored as a new version (fig.6.6). This shipment from the client to the file-server occurs when the user closes the file. Modifications to the file can only take place, when the user has the right to do this and when there is no lock on the file. The database takes care of the updating the meta-data of the file (the dataset). The file transport claims a lot of network bandwidth. Therefore it is advisable to place file-servers at all sites.

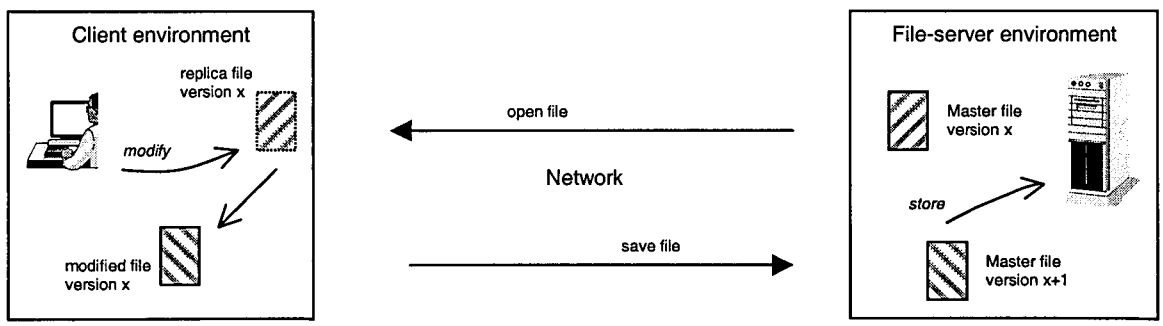


fig.6.6: file transport client - server

Management of meta-data of the files can be done by one central database system or by a distributed database system. The object (type dataset) representing a file will contain the address on where the file is stored (and the addresses of the replicas). It doesn't matter if this object is managed by a distributed database system or by a central database system.

Allocation Files

The files will be fragmentised where they are used the most. The *master* file will be placed on the site where user is located with access-rights to modify the file. Or in case multiple users at multiple sites can modify the file, the master file can be placed on the site, where the last modification occurred. There will be only one master file and only one user at the time can modify

[7] Size of the CAD files varied between 0.5 MB to 3.0 MB (small evaluation files stored in IRIS)

this master file. It is likely that files will be more viewed than modified. Replicas of the master can be distributed to other sites, therefore each file server will store *masters* and *replicas*. The database keeps track on which file-server the master file is located and on which file-servers replicas of the master file are stored. When there is a request of a remote file, then the system first check if there is a replica of the requested file is available on file-server of the "requesting" site, if not, a replica of the master is retrieved from the remote site.
It should also be possible to take a copy of an existing replica on another remote site, if the replica is up to date. Intelligence can be built in the system, that the system determines itself where it retrieves the file, dependable of the network performance. If the network connection with the site, containing the master, is slow then the system can look for a replica on another site with a faster network connection and retrieve a copy from that one.

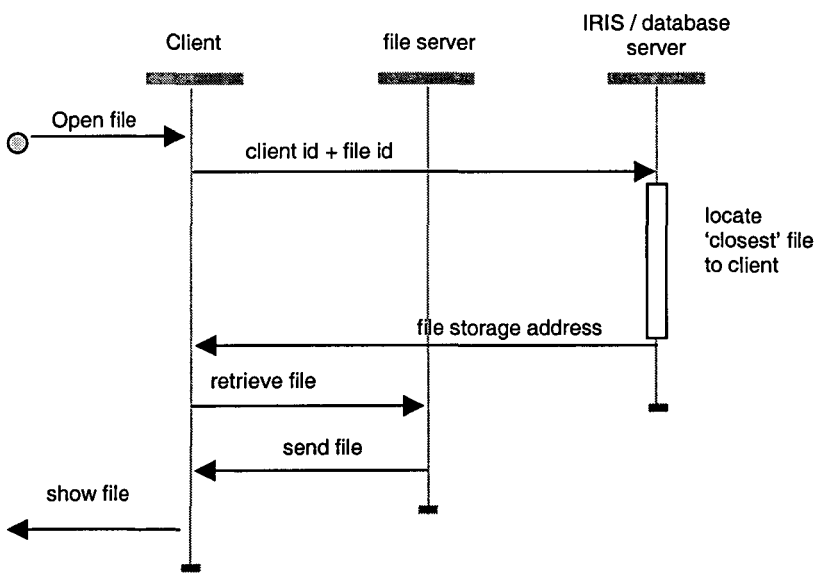
The file-server stores:
- masters: the original files
- replicas: copies of the original files of remote sites

Communication client ↔ database ↔ file-server

The database will act as 'co-ordinator' in the communication between client and file-server. The database is used for accessing the meta-data and locating address of a file, when the client wants to open a file then the transport of the file will be directly from the file-server to the client and visa versa for saving a file.

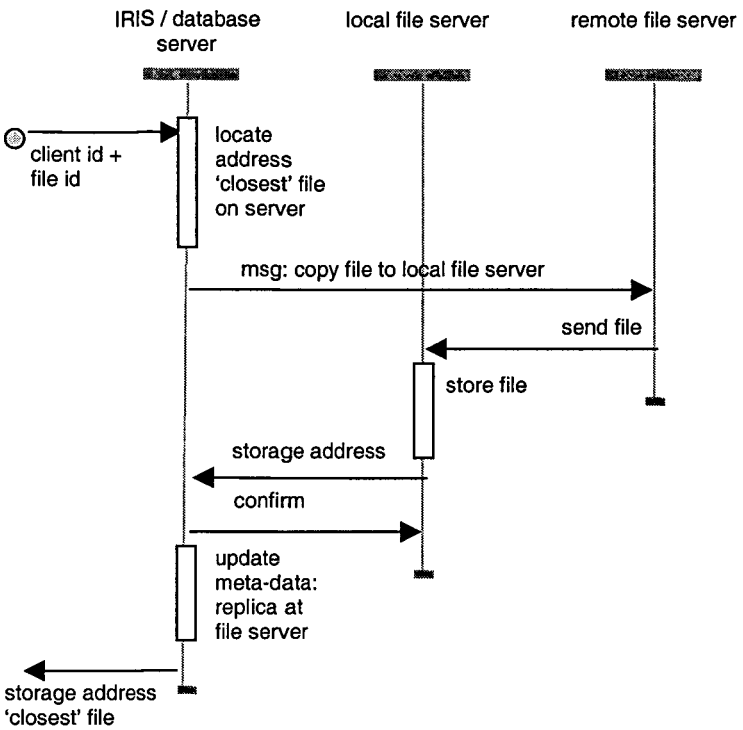
The following protocols give an illustration of the opening a file and storing a file. These protocols are simplified examples, other communications protocols are also imaginable. The application server IRIS and the underlying database are taken together.

Open file:



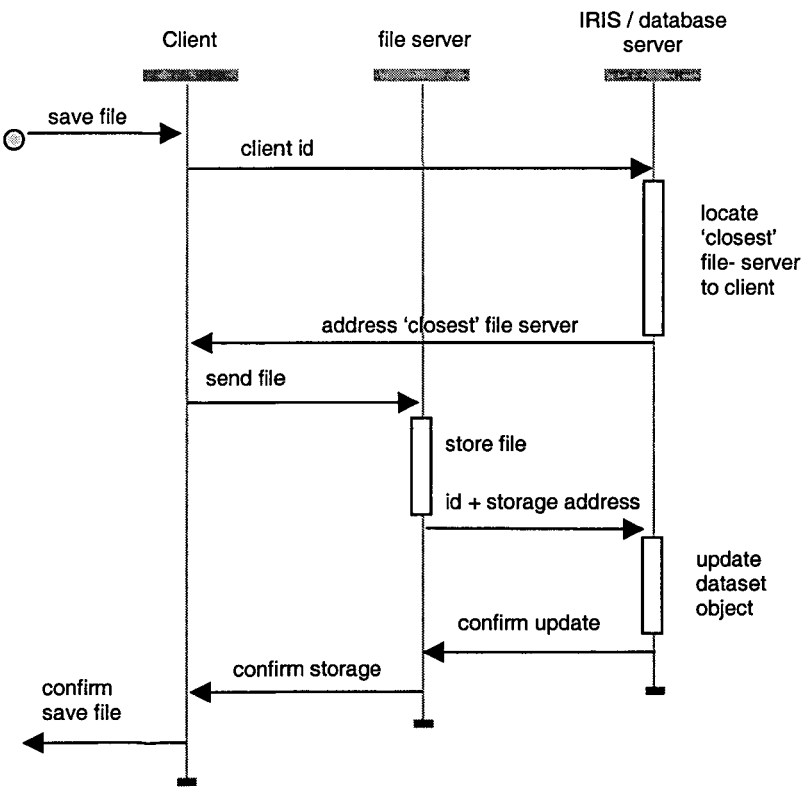
The user initiates the command *open file.* The command is sent to the IRIS database, where the database locates the file-server on which the file is stored. In case the file is not stored on the clients' site then first the file will be copied from the remote file-server to the clients' file-server (see next protocol). The database gives the storage address of the file back to the client, where after the client makes a direct connection with the local file-server to retrieve the file.

Copying file from remote file-server to local file-server:



The database registers that there is a replica of the file on the local file-server of the client. The next time the client opens the file, it can be directly retrieved from the local file-server (if the file is still up-to-date).
Saving a file can be on the following way:

Save file:



File Distribution Methods

Distribution of the replicas can be in several ways. There are two basic methods; *pull* distribution and *push* distribution. *Pull-distribution* means that a file will be sent to the site when there is a request for the file. With *push-distribution* the file will be sent to the site before there is a request.

The advantage of pull-distribution is that the transport over the global network is minimised. A file is only sent when there is a request for it. The downside is that the user has to wait longer before the file is on users' environment. Besides, it can happen that there are many requests on a particular time (for example at the beginning of the day), which can overload the network at that moment. The advantage of push-distribution is that the user has a fast access to the files. Replicas of remote files are already stored on the file-server, before the user requests for it. The disadvantage of this method is that it is unknown when the user accesses the replicas or if the user even needs the files. It can happen that the replicas will never be accessed, which results in unnecessary network transport.

- Pull distribution: the file will be sent on request
- Push distribution: the file will be sent before request

Pull & Push Distribution
Distribution methods can also be combined. There can be first pull-distribution and then after the first request push-distribution will be applied to the requested file. On this way only the files which are needed by the remote sites will be distributed, instead of just sending replicas to all sites. The protocol *Copying file from remote file-server to local file-server* is an example protocol of pull-distribution. After passing this protocol push-distribution for keeping the replica up-to-date can be applied. The push-distribution can be expanded with a timer, determining how long the push-distribution will continue. Perhaps files are only viewed for a short period. By expanding the push-distribution with a timer, the distribution will fall back to pull-distribution when the file is not viewed after a certain period. On this way unnecessary transport of files can be confined. Also the replica can be removed again from the file server after this period. Distribution can be real-time, new updates will be sent to the sites at the moment when a file is changed. The distribution can also be in defined intervals, for example every 24 hours.

Distribution in combination life cycle document
Distribution methods can also be coupled with the status of the documents and the workflow. For example, when a document has a provisional status then it can still change a lot. When modifications occur more than it is viewed by the remote sites, pull-distribution is more optimal. Or when a person on a remote site has to signoff a document, the file can already be pushed to the remote site when the release procedure is initiated. Push-distribution can be used between the sites, which are part of a product chain, while to unrelated sites pull-distribution is used. Many combinations are possible, dependable of the workflow of the file and the user behaviour.

The transport of the files has an impact on the network load, thus determining the distribution method is of a great importance. By combining distribution methods and building some intelligence into the system optimal distribution of the files can be accomplished. To receive the optimal method, the network and the user behaviour have to be monitored. The user behaviour can change in time, thus the distribution methods have to be flexible to adapt the changes.

6.6. Infrastructure: Network Requirements

Next to the performance of the database, the network has to be capable of handling the input and output of the system. There is no use to construct a very fast system when the network can't handle the intended communication. In this paragraph the basic factors that impact the network performance are described. Furthermore, it is explained how to estimate the network requirements for the proposed architectures in the previous paragraphs.

Network performance is measured in two fundamental ways: *bandwidth* (also called *throughput*) and *latency* (also called *delay*). The bandwidth of a network is given by the number of bits that can be transmitted over the network in a certain period of time. For example, a network might have a bandwidth of 10 million bits/second (Mbps), meaning that it is able to deliver 1250000 bytes every second. Latency corresponds to how long it takes a single bit to propagate from one end of a network to the other. Latency is strictly in terms of time. Latency is determined by three components: propagation delay, transmit and the queuing delays inside the network.

Latency = Propagation + Transmit + Queue
Propagation = Distance/SpeedOfLight (constant)
Transmit = Size / Bandwidth ('Size' is the size of the sent packet)

The word *throughput* is used to refer to the *measured performance* of a system. Thus, because of various inefficiencies of the implementation, a pair of nodes connected by a link with a bandwidth of 10 Mbps might achieve a throughput of only 2 Mbps. This would mean that an application on one host could send data to the other host at 2 Mbps. The *bandwidth requirement* of an application is the number of bits per second that needs to be transmitted over the network to perform acceptably.

Network performance versus database architecture

The choice of the database architecture is mainly determined by the performance requirements of the system. The limiting factor of a central database system is the network capability. The network requirements for both architectures can be estimated by measuring the data flows generated by transaction processing (central system), and data flows generated by replication activity together with global transactions (distributed system).

In order to determine if one central database system is technically possible with the current network, it has to be measured how much data transport takes place between the clients and the server. This amount of data transport has to be split up in meta data and physical data. The amount of meta data has to be compared with the throughput of the network between the database site and all the remote sites. For meta data a constant (real-time) connection is required. Physical data can be distributed in intervals, for example in the moments when the network load is not so high.

remark:
The global network is already used by other systems. A 2 Mbps connection with another site, doesn't automatically mean that 2 Mbps bandwidth is available for the system IRIS. Probably a large part of this 2 Mbps is already 'occupied' by other systems and data transport.

Because of the lack of monitor tools and a good description of the communication between the client and the server, it was not possible to give an accurate estimation of the network load of a transaction between a client and the server. In order to give an impression of the network load of both architecture concepts (central or distributed), assumptions of the communication between client end server are made.

Transaction Client - Server
A transaction will be defined as a request of the client to the server followed by the reply of the server to the client (fig.6.7). The performance of the infrastructure can be assessed by the time it takes to perform a transaction. The transaction time is the time between starting up the request by the client till the time the client receives the reply.

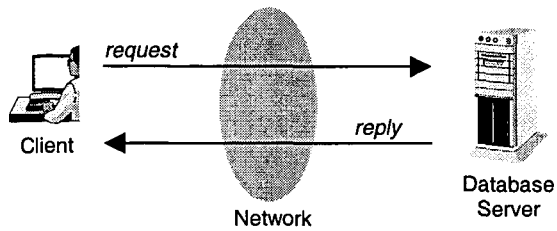


fig.6.7: Transaction client-server

The transaction time can be split up in 5 parts:
1. computing time client
2. latency request
3. computing time server
4. latency reply
5. computing time client representation reply

- transaction latency

The computing time of the client and server can be considered the same for both architectures. The infrastructure aspect of the transaction time is the latency of the request and the reply.

latency request = propagation + request size / bandwidth + queue client-server
latency reply = propagation + reply size / bandwidth + queue server-client

The *queue client-server* and the *queue server-client* together with the propagation delay is the time of sending a package from client to server and back (two-way latency). The *Round Trip Time (RTT),* also called the *ping,* can be used as an estimation of the queuing time and the propagation delay. The RTT is the time it takes to send one bit from one end of the network to the other and back. The network latency of the total transaction (request and reply) can be simplified to:

transaction latency = (request size + reply size) / bandwidth + RTT

The *request size* and the *reply size* depend of the kind of transaction. Most of the transactions will be for retrieving information of the database. Retrieving information can be by a find query or by 'walking' through the directories and/or product structures. The size of the reply can vary with each transaction. Furthermore, the occurrences of the transactions are important; the number of transactions performed by users at the same time. 'Walking' through a product structure, for example, is established by many transactions behind each other requesting the next level in the structure. By taking all the parameters, an estimation of the transaction latency can be made.

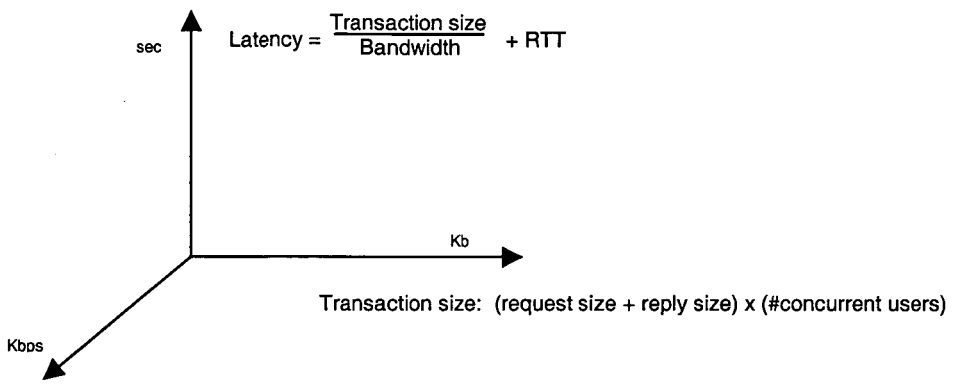
Example:
10 concurrent transactions per sec. (10 users performing one transaction at the same time)
request size: 1 KB = 8000 bits
reply size: 15 KB = 120000 bits (average size of web page measured with iMAN Web)
Bandwidth from Eindhoven - Brazil: 512Kbps
RTT Eindhoven – Brazil: 500msec

transaction latency = (8 + 120) x10 / 512 + 0.5
= 3.0 sec.

Lowering transaction latency can be done by increasing the bandwidth. Taking the same values of the previous example, but then with a bandwidth of 2Mbps gives a latency of 1.14 sec.

transaction latency $= (8 + 120) \times 10 / 2000 + 0.5$
$= 1.14$ *sec.*

The following model can be used to determine the network bandwidth.



Bandwidth: dedicated throughput of the network for IRIS

To give an accurate estimation of the required network bandwidth for a central database system, it is necessary to investigate the user behaviour and determine the size of transactions. This can be done by measuring the current data flow between the client and the database server of the current system on the local area network. Take the periods when the system is used the most (e.g. beginning of the day, just before and after lunch) and calculate the data transmitted per second. Hereby, it has to be taken into account that when the number of users grows, the bandwidth need to increase too. Furthermore, the desired transaction time has to be set (*performance goals*) and to calculate the maximum acceptable latency. Taking the two parameters, the required bandwidth can be deducted.

For a distributed database system the (global) network bandwidth requirement is smaller than for a central database architecture. The global network will be mainly used for updating the replicas. Updates only occur when data is modified, however, reading data will occur much more than modifying data. Furthermore, the size of the transaction for an update will be smaller, because probably only a few objects will be updated.

See Appendix V for bandwidth and Round Trip Times of the global network.

7. Designing *One Logical* Database

In the previous chapter, the two concepts of database architectures for storing meta-data are described: a central database system and a distributed database system. In a distributed database system, the meta-data is distributed on several computers on multiple sites. A distributed database system appears to the user as a single database but is, in fact, a set of two or more databases, which are joined together forming *one logical database* (fig.7.1). In order to form one logical database, (representatives of) the objects will be replicated and stored at each site. The changes have to be made to relate the objects of one site with objects of another site.

In this chapter it is described how to create one logical database. First, the concept of relating the separate parts of the product structure is explained. Next, changes to the current data model of the product structure are introduced in accordance with the requirements stated in chapter 4. In paragraph 7.3, it is explained how these changes can establish a shared product structure by sending representatives of the objects to all the sites.

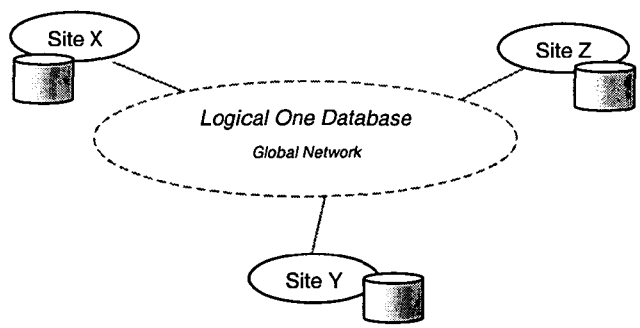


fig.7.1: A distributed database: meta-data stored on multiple locations

In a central database system no changes to the data model have to be made (assuming a correct data model). The meta-data forming the structure is stored in one database, which already provides a central view of the product data.

7.1. Approach

There are two major design approaches for designing a distributed database: the top-down and the bottom-up approach. Like the names already indicate, one approach starts at the top looking at the user requirements and starts a conceptual design without taking into account the distributed aspects at that point. Top-down design is a suitable approach when a database system is being designed from scratch. However, multiple installations of the PDM-system IRIS are already operating at the sites, the design task involves integrating them to one database. The bottom-up approach is suitable for this type of environment. The starting point of the bottom-up design is the individual conceptual local schema (local level). The design-process consists of integrating all local schemas of the individual sites into the global conceptual schema, which describes the logical structure of the global data of all the sites (global level).

All installations of the PDM system IRIS at the sites are the same, the environment of is homogeneous. Thus, there is only one local conceptual schema, which is used on all sites. Besides, the enterprise view of the product structure on the local conceptual schema is similar as

the view on the global conceptual schema. Only on each site different product data is stored (excluding the replicated product data). Currently, separate parts of the structure are stored on multiple sites without any relations to each other. The user wants to view the whole product structure transparently of the location where the parts are stored. In order to make an overall view of the product structure, the parts of the structure have to be connected with each other (see figure 7.2.).

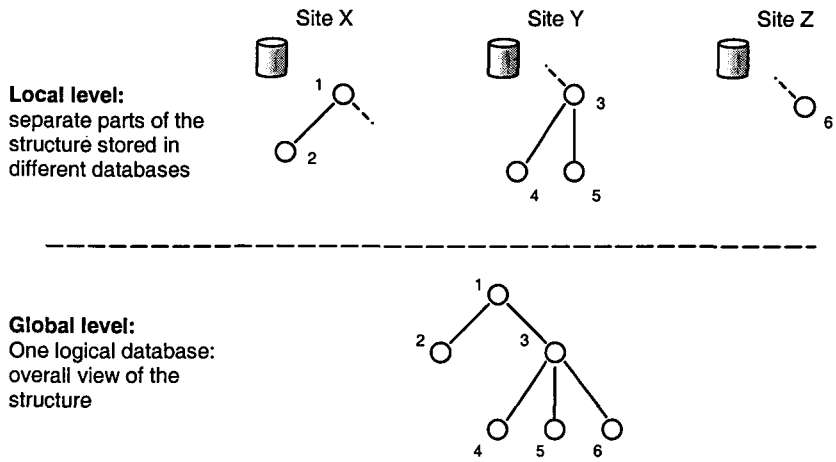


fig. 7.2: Relating the parts of the product structure on global level

To accomplish one logical database, the object model of the product structure has to be changed, so that it becomes possible to relate data of multiple sites. The principle behind relating the data is to introduce *relationship* objects, which connects one *data* object with the other. The data objects themselves will not be changed, but a new *relation* object will be created to link the data objects together. It doesn't matter whom or which site owns the data object.

In the next paragraph the necessary modifications of the data model are described. In this chapter data on *local level* is only accessible by the local site. Data on *global level* is accessible by all sites.

Remark:
The ODS (Object Directory Service, described in chapter 5) stores only the parts of the structure of each site separately. Looking at figure 7.2, this means that the ODS stores only separate parts of the local level.

7.2. Global Data Model of the Product Structure

In this paragraph changes to the current data model of product structure (described in chapter 3) are introduced in order to establish a shared product structure. According to the user requirements, described in paragraph 4.4.1, all (global) users have to be able to view, create and modify the product structure. First, it is explained why the current data model is not suitable for relating one data object with another. Second, modifications of the data model are introduced for relating objects.

The following class-diagram can be made of the existing product structure (fig.7.3). This is the same model as given in chapter 3, but represented in a slightly different form. Class *Form* is for a

moment neglected, because it only adds some additional attributes. (In appendix I a clarification of the class-diagram is given.)

remark:
In order to make it more readable the following designation will be introduced:
example: *an object/instance of the class Dataset* will be abbreviated to *dataset*

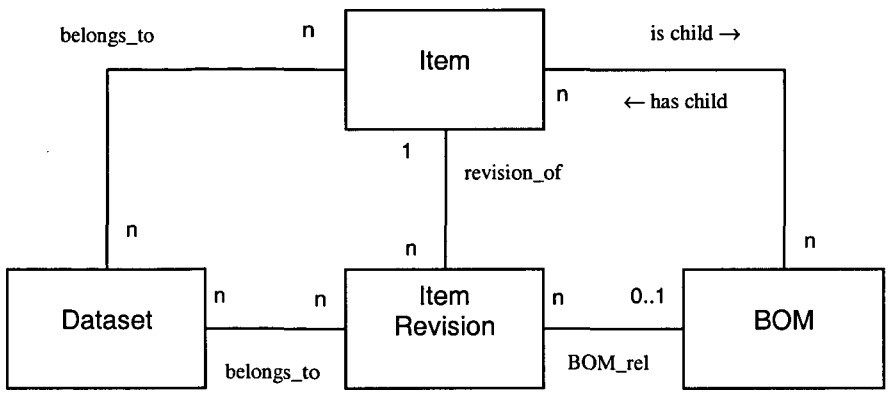


Fig.7.3: class-diagram product structure

In this model it is assumed that the existing datasets are only connected with an item or item-revision. It is possible that a dataset is attached to multiple items or item-revisions. In practice a dataset could also be connected to an object of the type Folder. In this report only the data model of the product structure will be specified. The product structure in relation to folders or other structures can be handled on a similar way.

The classes contain references to each other to establish the product structure. Here lies the problem. When a user wants to relate an object with another object, the object (meta-data) has to be altered. Suppose the following implementation of class Item. The class Item has probably a set of references to its revisions and a set of references to the datasets, which are attached to the item. All attributes, like *owner*, *status*, etc. are summarised for the moment in *additional attributes*.

class Item
id: *Item_id*
revisions: *Set of Revision_id*
datasets: *Set of Dataset_id*
add.attributes: *value*

Where Revision_id is the identifier of an object Item-Revison, Dataset_id is the identifier of a dataset.

Creating a new item-revision is not only creating a new object of the class Item-Revision, but also connecting it with the item of which it is derived. The item in question has to be modified by adding the *revision_id* of the new created item-revision to its set of revision-ids:

revisions := revisions ∪ { 'new revision_id' }

Connecting a dataset to an item goes on the same way:

datasets := datasets ∪ { 'dataset_id' }

In a central database scenario, the site owns all the objects of a product structure. All the objects are stored on one site in one database. Changes in structure (adding data, new BOM, etc.) can easily be done by altering the object concerning the relations of the structure. With multiple databases, the objects of a product structure are distributed among the sites. An object is stored at and owned by one site. The separately stored objects have to be connected to construct a product structure.

The objects published on the global level have to be site-independent and 'free' to enable a global user to construct product structures or to add new data to existing data. To maintain the autonomy of the owner of an object, an object must not be altered when it is used on global level in a product structure. Therefore, the connections between the objects have to become independent of the object on global level. For adding data to existing global data or constructing a product structure, relationships have to be made between the objects.

The relations 'implemented' in the classes have to be disconnected from the classes and kept registered separately. All methods for establishing a relation between two objects will be kept in a relationship-class. The relationship *belongs_to* between the objects of the class Dataset and the class Item will become a *Belongs_to* object (fig.7.4).

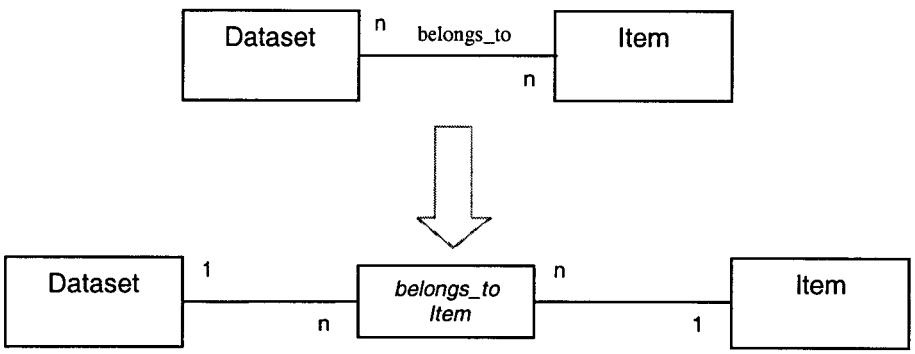


fig.7.4: Relation class: belongs_to_Item

Class belongs_to_Item can be seen as a pair *(Dataset_id, Item_id)* representing a connection between the two objects. For every connection between a dataset and an item a *belong_to_item* object is created.

class *Belongs to Item*
id: *id_value*
from_dataset: *Dataset_id*
to_item: *Item_id*

The variable *from_dataset* is the identifier of the attachable dataset and the variable *to_item* is the identifier of the item to which the dataset is connected.

The class Item contains no longer the sets of references and contains only an identifier and the additional attributes.

class *Item*
id: *Item_id*
add. attributes: *value*

Those relationship classes can also be introduced for the other relationships between the classes in figure 7.3.

class *Belongs to Item-Revision*	
id:	*id_value*
from_dataset:	*Dataset_id*
to_item-rev:	*Revision_id*

class *Revision*	
id:	*id_value*
master_item:	*Item_id*
revision:	*Revision_id*

class *BOM_parent*	
id:	*id_value*
parent:	*Revision_id*
children:	*BOM_id*

The classes Item, Item-Revision, Dataset contain now only information about themselves and don't contain any references to other objects anymore. Class BOM is already a relation-class, containing a group of relationships of which the parent (an item-revision object) is the same. However, an item-Revision doesn't have to contain a bill of material when it is created. At a later stadium a bill of material (object BOM) can be attached to the item-revision object. So, also between the class Item-Revision and BOM a relationship-class is introduced. It is also possible to replace class BOM of the form *(Item-Revision_id, {Item_id}).*

The modified class-diagram will be:

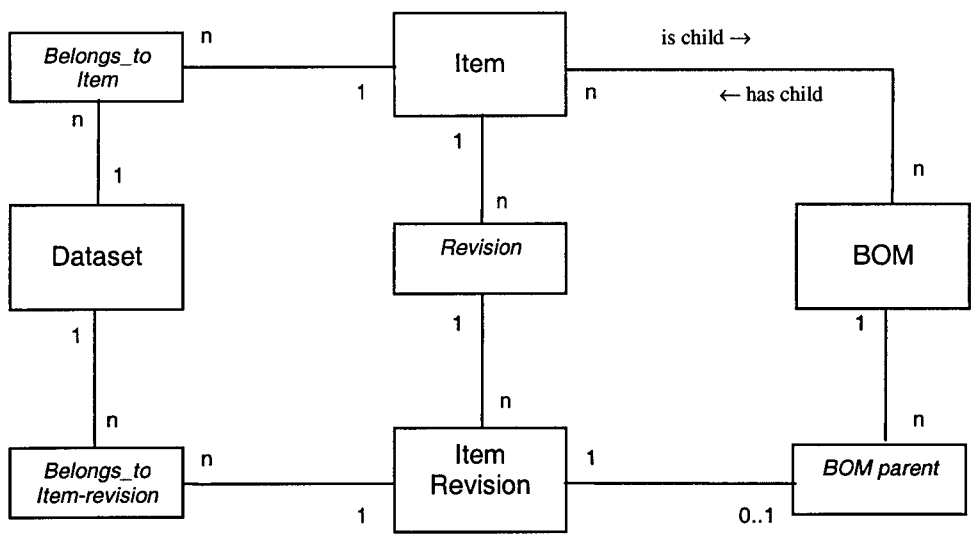
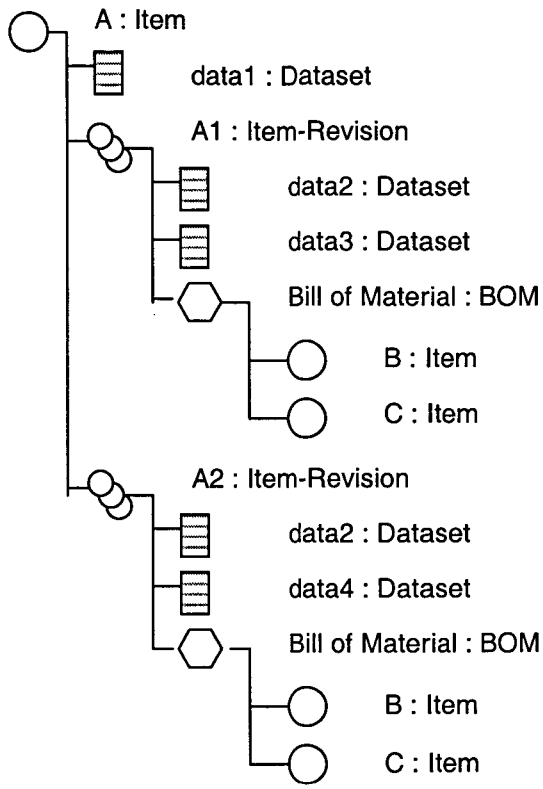


fig.7.5: Modified class-diagram product structure

Making relationships between two objects can now be done without modifying them by creating a relation-object. The relation-objects together with the BOM objects are forming the skeleton of the product structure.

The following example of an item, consisting of two sub components, shows how the modified data model of the product structure looks like. Instances of the classes can be represented in a table form.

Example:



Dataset	
Data-id	*Attributes*
data1	...
data2	...
data3	...
data4	...

Item	
Item-id	*Attributes*
A	...
B	...
C	...

Item-Revision	
Revision-id	*Attributes*
A1	...
A2	...
B1	...
C1	...

BOM	
BOM-id	*Item-child*
b1	{B,C}

Relation-object tables:

Belongs to Item		
id	*from_dataset*	*to_item*
1	data1	A

Belongs to Item-Revision		
id	*from_dataset*	*to_item-rev*
1	data2	A1
2	data3	A1
3	data2	A2
	data2	A2

Revision		
id	*master_item*	*revision*
1	A	A1
2	A	A1
3	B	B1
4	C	C1

BOM parent		
id	*parent*	*children*
1	A1	b1
2	A2	b1

remark:
Other data models of the shared product are also possible. The described data model is an example for showing how objects can be related to each other in order to form a shared structure.

7.3. Creating a Shared Product Structure

The objects have to be visual at all sites in order to create a shared structure. The data dictionary catalogues all objects in the global scheme. The challenge here is how to make the dictionary available to and identical at all sites. A centralised catalogue means that the whole system relies on one single point. If this point falls away then all the 'connections' between the sites will fall away. Another option is to make each site responsible for its portion of the catalogue only, although this is not very practical since resolving the location of remote objects would launch a blind query to all the remote sites in the search for the referenced objects. This is very time consuming and will definitely lower the performance of the system. Another solution of this problem is to store information about the location of remote objects in each local data dictionary. It means that there is no such thing as a global data dictionary in the pure sense, but on all sites the locations of all published objects are stored. Unlike the ODS, which is a separate database (centralised catalogue), every site will have its own catalogue stored in its own database.

The amount of information available about the remote objects varies from one database to another, but at the very least the identifier and the physical location of the objects have to be recorded. Besides, it is also desirable to include some additional information about the remote objects, for example about the distribution, so that the system can optimise queries effectively. The amount of information and the distribution of the object also depends on the network performance. It has to be balanced between the minimum information necessary for viewing the structure and the capability of the network. For a shared product structure the following information of objects has to be available on each site:

- the published object id's (id contains the location of the object)
- the published relation-objects
- the published BOM objects

Making information available for other sites
Suppose that on every site there is a location where the user can access the published objects of a product. With a remote query a project folder can be created on all sites with a link to the product information. When a site wants to share objects with other sites, representatives (replicas) of the objects will be sent to the other sites. These representatives will be stored in the local database of the sites, where the user can view them. The *replica-objects* can only be viewed, not modified, but relationships between the objects can be made. Relationships are objects too, so making a relationship between the existing objects, means creating a new 'relationship' object.

A site can send first notifications of the available product data to all remote sites, for example only the root or the skeleton of a product structure (push distribution). When a remote site wants more information, it can get a subscription of the whole product structure or a part. Several distribution methods are possible (similar to those explained in paragraph 6.5.). Release procedures, for example, can be combined with the distribution of the product data to the right persons. When a document is released to a certain status, it can automatically be distributed to other sites. The distribution methods can be combined with the whole workflow. Distribution methods however have to be sealed off from the user.

7.3.1 Fulfilment User Requirements

The fulfilment of the user requirements written in paragraph 4.4 is as followed:

Retrieve information

At every site, the local database contains master objects and replica objects (meta-data). The objects of the global product information are replicated to all concerned sites. Retrieving the global product information is similar as retrieving the local information of the site. The user will not notice that the meta-data stored at the database of the product contains replicas or masters. The meta-data can be accessed on the same way as the current system, using a find query or using the directory mechanism.

Publish information

Publish stand-alone data
Stand-alone data can be seen in a distributed database environment as data owned by one site, which has no relation with data of other sites. Making this data available for others is easy, (replicas of) the objects of the stand-alone data will be sent to the concerned sites.

Add new data to an existing component
It is given the existing data of a component, which is an item object and/or item-revision objects of the component. These objects are known to the user. Adding new data to the component can be adding a dataset object and/or an item-revision object.
In order to add new data to the existing data, the new data objects (datasets and/or item-revisions) first have to be created. Next, these new data objects have to be related to the existing data by creating *relationship* objects (*belong_to_Item(-Revision), revision*). The new created objects will be sent again to the concerning sites, so that also the other sites will see that new data is added. The existing objects stay unchanged.

In case the 'new' data already exists then relating it to other existing data is just creating a relation object to connect the two objects. In practise this occurs for example with using the commands *copy & paste.* A user copies a dataset from one location and 'pastes' it into an Item.

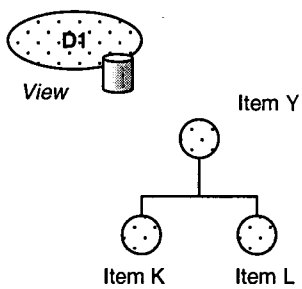
Adding a new BOM
Adding a BOM to a leaf node of the product structure or modifying an existing BOM happens by creating a new BOM object. Creating a new BOM is almost similar as the previous operation. Select the children (*Items*) and group them together by creating a new BOM object. Then relate the new BOM to the parent (*Item-Revision*) by creating a relationship object (*BOM_parent*). By sending these objects to the other sites everybody will see that there is a new Bill of Material created.

Constraints:
Global constraints have to be built in to assure that the data stays consistent and orderly. For example rules for creating new item-revision: when a site creates a new revision of an item, then what to do with the datasets attached to the previous revision? Do these datasets automatically have to be included in the new-revision? Will the previous revision become frozen (read-only) automatically? Probably much more rules have to be specified in order to have a consistent and orderly product structure. (These are also issues for a central database concept.)

The principle of establishing a shared product structure is illustrated in the next two scenarios.

Scenario Shared Assembly:
Assume that 2 design sites *D1* and *D2* are working on one product *Y*. The product consists of 3 parts K, L, M. Site D1 owns the root of the product and component K and L (Items Y, K, L). Site D2 designs component M. *D1* starts the design of product Y and creates *Item Y, K, L* and for each item the first Item-Revision *Y1, K1, L1*. With a BOM object the item *K* and *L* are related to Item-Revision *Y1*. Site D2 designs component M and wants to relate item M to the rest of the product structure (see figure 5.3 in chapter 5).

Create first product data



Item (site D1)	
Item-id	*Attributes*
D1.Y	...
D1.K	...
D1.L	...

Item-Revision (site D1)	
Item-Rev.-id	*Attributes*
D1.Y1	...
D1.K1	...
D1.L1	...

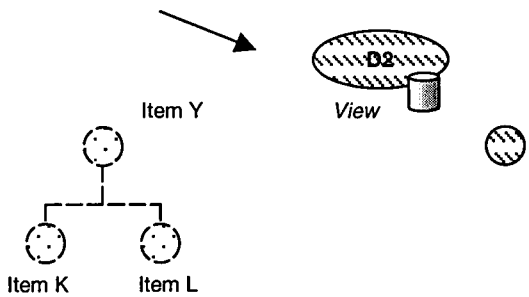
BOM (site D1)	
BOM –id	*Item-child*
D1.b1	{ D1.K, D1.L }

BOM Parent (site D1)		
id	*parent*	*children*
1	D1.Y1	D1.b1

Site D1 publishes structure

Items: { (D1.Y, ...), (D1.K, ...), (D1.L, ...) }
Item-Revisions: { (D1.Y1, ...), (D1.K1, ...), (D1.L1, ...) }
BOMs: { (D1.b1, {K,L}) }
BOM-Parents: { (D1.1, Y1, b1) }

Create Item M

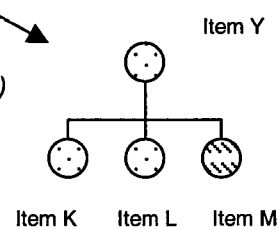


Item (site D2)	
Item-id	*Attributes*
D2.M	...
...	...

Item-Revision (site D2)	
Item-Rev.-id	*Attributes*
D2.M1	...
...	...

Link Item M to structure

On site D2:
Create new Item-Revision (D2.Y2, ...)
Create BOM object with children (D2.b1,{D1.K,D1.L,D2.M })
Create BOM-Parent (D2.1, Y2, b2)



Site D2 publishes item M in relation with structure

Site D1 receives the added data of site D2 and stores this in its own database. On both sides the following objects are visual:

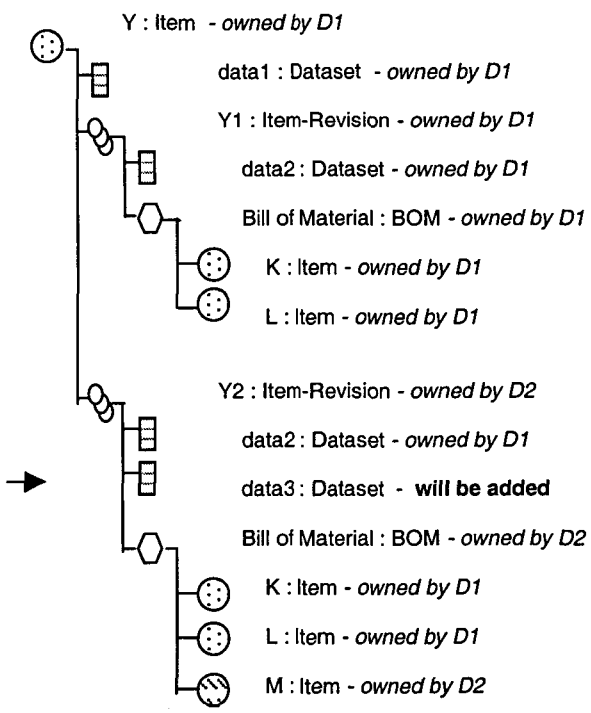
Item	
Item-id	*Attributes*
D1.Y	...
D1.K	...
D1.L	...
D2.M	...

Item-Revision	
Item-Rev.-id	*Attributes*
D1.Y1	...
D2.Y2	...
D1.K1	...
D1.L1	...
D2.M1	...

BOM	
BOM –id	*Item-child*
D1.b1	{ D1.K, D1.L }
D2.b1	{ D1.K, D1.L,D2.M }

BOM Parent		
id	*parent*	*children*
D1.1	D1.Y1	D1.b1
D2.1	D2.Y2	D2.b1

Scenario Shared component:



Taking the assembly of theprevious scenario, component Y looks like beside. There are 3 datasets of site D1 attached to the item. Site D2 wants to add a new dataset (*data3*) to revision Y2.
Adding a new dataset goes on the same way as adding a new BOM:

Site D2:
- *create new Dataset : (D2.data3, ...)*
- *create new Belongs_to_Item-Revision : (D2.r1, D2.data3, D2.Y2).*

The new items will be distributed again.

Dataset	
Data-id	*Attributes*
D1.data1	...
D1.data2	...
D2.data3	...

Belongs_to_Item (-Revision)		
id	*from_dataset*	*to_item(-rev)*
D1.r1	D1.data1	D1.Y
D1.r2	D1.data2	D1.Y1
D1.r3	D1.data2	D2.Y2
D2.r1	D2.data3	D2.Y2

7.3.2 Fulfilment System Requirements

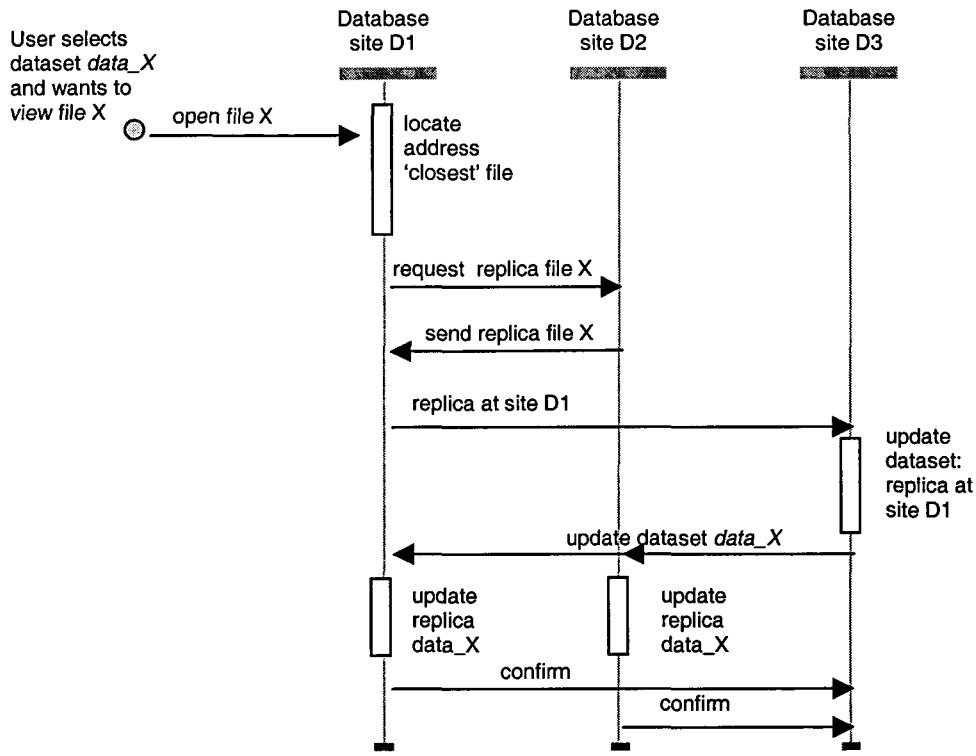
Preserving the integrity of the data

The system has to make sure that the data at all sites is correct and up-to-date. All the users have to view the same data. In order to accomplish that, the objects are replicated and sent to all the sites. When modification to the data are made, then the replicas of the objects have to be updated.

Updating objects
An update of the replicas occurs when the master object is modified. In most cases object dataset will be updated, because the file, which the dataset represents, will be modified. But also, for example, when an object is released to a new status, the replicas have to be updated with the new status. Updating the object-replicas can be done in real-time or in intervals.

Updating datasets
Updating an object of type Dataset can be better combined with the updating of the physical file, which the dataset represents. If a new dataset object is created, it is not a problem to distribute it real-time. For updating a dataset object, however, the same method has to be used as updating the physical files. It has no use to let sites know that a file is updated by sending an update of the dataset of the file, while the replica of file will be replaced at a later time.

In case the distribution method is chosen that also allows that copies of replicas can be taken (see paragraph 6.5), then the object dataset also have to be updated when a replica file is taken of and stored at one of the sites. Suppose site *D1* retrieves a replica of file *X* of site *D2*, and site *D3* is owner of the master of file *X*. The dataset representing file *X*, assume named *data_X*, has to be updated with the fact that site *D1* has a replica. This means that dataset *data_X* owned by site *D3*, has to be modified, because of an action of site *D1*. All the object-replicas of dataset *data_X* have to be updated too.



remark:
In case only replicas can be taken from the master, then the distribution information of the replicas (meta-data) doesn't have to be replicated to all the other sites. Only the site, containing the master file, keeps track which sites have replicas. Depending of the network infrastructure, it can be decided which method to use. For example, if in the USA many sites need to view the same files, of which the masters are stored in Europe, then it is advisable to take copies of replicas already stored in the USA instead that every site retrieves their own replica of Europe. (The bandwidth available between Europe and USA is not so high, see appendix V.)
With a central database all the meta-data is stored on one place, updating the replication information is not an issue. Taking copies of replicated files is very easy to implement.

Last modified date
Furthermore, rules have to be made and implemented in the system to deal with the attribute storing the modification date. Taking the previous scenario *shared component* (in paragraph 7.2.1), the site D2 adds a dataset to the Item-revision Y2. This means that the content of Item is modified, so when a user views the item Y then the attribute *last modified date* has to be set with the date when the site D2 added the dataset (assuming this was the last action). A simple rule can be that the *last modified date* of an item is the latest *last modified date* of the objects attached under the item and under the item-revisions. Applying this rule to the object item is an *external view* of the stored data.

Failure of a database
Logs of all changes to the data have to be made, in case of the one the sites fall away. When the failure is fixed then the logs have to be applied to the upcoming site, so that the upcoming site is up to date again.

Uniqueness objects

It is stated that item objects are unique. In IRIS, it is not possible to have two distinct items with the same name. For product data this uniqueness holds for the item with a 12NC identifier. The 12NC's are extracted from the system PRIME. The system IRIS doesn't provide a 12NC, the identifier has to be filled in by the user. IRIS only checks if the12NC already exists in the database. A mechanism has to be developed to assure that the identifiers (12NC) are unique at all sites. An easy way to assure this can be by launching a blind query to all the sites and checking if a 12NC already exists, when a new item object is created. This can be very time consuming, although creation of a new 12NC doesn't happen many times. Currently the system, assures that all items are unique, also items with a non-12NC identifier (this is very easy at a central system). The uniqueness of items can be restricted to the uniqueness of items of official TPI, which are items with a 12NC.
It is perhaps also possible to integrate the systems PRIME and IRIS, so that PRIME denotes 12NC identifiers to item objects.

Datasets however can have the same identifier. In order for the system to distinguish the objects with the same identifier, it uses name-synonyms. The user sees the identifier which the user gave to the object. The system uses an unique identifier as synonym for the object. The uniqueness of this synonym can be easily assured in a distributed system by adding to the identifier the unique site name.

Example unique identification:

site identifier :	*D2*
users' view dataset object identifier :	*document1*
unique internal identifier (synonym) :	*122123*
global identifier :	*D2.122123*

Local – Global data

Adding local data to global data goes on the similar way as illustrated in the scenario, only the added data will not be distributed. Also here, it has to be investigated what to do with the *last modified date*. Adding local data to an item means that for this site the item is changed, but for other sites the item remains the same. So, a *local external view* of an item can be different than the *global external view*.

Performance, High Availability, Scalability

Performance, High Availability and Scalability are provided by the architecture of the distributed database system (see chapter 6).

8. Recommendations

In this report, the distribution aspects and the conceptual design of a global PDM system are focussed on the product structure. The basic functionality for viewing, creating and modifying the product structure are defined and worked out (chapter 4). The product data has to be accessible by all the sites. At the moment multiple PDM systems are operating independently of each other at various sites. The product data stored in the systems are not related with each other. The PDM system IRIS doesn't provide the functionality, which is necessary for product structure distribution. In this report, conceptual solutions are given, which provide global accessibility of the product data. The following conclusions and recommendations can be made:

- An examination of the current data model of the product structure concludes that the structure is not consistent. The data model of the current product structure is too open. The user has too much freedom to create all kinds of variants of the structure, mainly because of the wrong configuration of the relationships between the objects. The structure is well defined on paper and the users are supposed to keep to the rules, so why not implementing these rules as constraints into the structure? Before introducing Release & Change procedures into the system, the relationship errors have to be fixed. It has no use to secure data with procedures, while everybody can make and delete links between files (see paragraph 3.2.2).

- The product structure can be described in a relative simple data model, but in the near future the structure can become more complex. At the moment the design information and the manufacturing information are stored together in one structure. It is imaginable that there will be needs to store this information in two separate structures (one for design and one for manufacturing), which also need to be related to each other. Furthermore, the process data is also stored in a (process) structure, which has to be related to the product data. Integrating all structures makes the data model more complex and more complicated. Therefore it is important to have a solid foundation for data storage on which data models can be designed, created and altered

- The distribution facility ODS offered by IRIS for sharing data between sites is not suitable for the desired way of working. With the ODS it is only possible to distribute 'stand-alone' data: data which has no relation with data of other sites. Looking at the information flow of the future, it is not recommended to implement this facility (see chapter 5).

In this report two conceptual architectures for data storage are described. Deducted from the user requirements there is a need for a central view of all product data. Next to this, the two important requirements for the architecture are *scalability* and *high availability*:

- *Scalability:* The system is still in the introduction phase, with a relative small number of users. When the system becomes a success, the number of users will increase and the data stored and managed by the system will increase as well. Therefore the system has to be scalable in number of users and data storage. Next, new sites will emerge or will be shifted to other locations, the system has to be able to adapt to the changes in the production facilities.

- *High Availability:* Sites are located all over the world. Because of the different time zones, the system has to be operational 24 hours. Furthermore, all product and process information is managed by the system. Downtime of the system means that 'the gateway' to the information is closed, resulting in slowing down the development and manufacturing progress.

The proposed architectures, a distributed database system and a central database system, can provide both requirements (explained in chapter 6). In the architectures, a distinction between the

meta-data and the storage of files is made. With a central database the meta-data is stored on one location in one database, which handles the requests of all 'global' users. The files will be stored locally at the sites, to lower the network load of transporting the relatively large files over the global network. The concept of a distributed database system is to have multiple independently operating databases, communicating with each other to form *one logical* database. The meta-data together with the files are distributed among the sites. The requests of the clients are 'all' handled locally at the local network. The network requirements for handling the global requests with a central database still have to be investigated. It also has to be taken into account that, with a central database, an increase of users results in an increase of the network load. With a distributed database system this increase of network load will be less.

- The physical data flow between the clients and the database server still has to be investigated in order to determine the network requirements (paragraph 6.6).

The decision, which architecture to take, will probably be based on the costs. A distributed database system is more complex than a central system, thus the costs to develop the system will be higher. However, for both architectures adjustments have to be made to the current system, although for a central database system the modification are probably lesser.

- The user interface of the PC version of IRIS is not the friendliest one. Many unknown error messages appear on the screen and the stability needs some improvement. PDM is a new technology and implementing a new technology in a large organisation is a long process of convincing everybody of the benefits. However, with a robust and user friendly system the first steps in convincing are already taken.

- Next to the improvements to the client side, it is also advisable to install some tools to better monitor the system, so that the system can be better configured to the users' behaviour.

- The web version of iMAN seems to offer global accessibility, but it is still steps away from constructing a global product structure. The system only offers a remote login for viewing the information through a web interface. The top layer of the system (the interface) is changed and improved, but the architecture and the functionality stay the same.

The PDM system IRIS is a variant of the PDM system iMAN, which software company Unigraphics Solutions offers. IRIS is a turnkey product delivered to Display Components. Display Components can define it's own business rules and can determine itself how the PDM philosophy will be implemented. However, the design and the 'production' of the system are in the hands of UG Solutions.

The current PDM system iMAN doesn't provide global accessibility of the product data on the way Display Components desires. In this report, conceptual solutions for providing the global accessibility are described, but implementing the solutions results in altering the current design of the system. It still has to be investigated how flexible the design structure of the current system is. Perhaps the original vendor delivered the system in such a way that modifications of larger scale are not possible or the costs will be very high. If this is the case, it is perhaps advisable to investigate, if it is still profitable to modify the current system or to look out for alternative systems.

Bibliography

Cooper R. *"Object Databases: An ODMG Approach"*, International Thomson Publishing Inc., Boston, 1997.

Dye C. *"Oracle Distributed Systems"*. O'Reilly & Associates, Inc., Sebastopol, 1999.

Hee v. K.M. *"Information Systems Engineering: A Formal Approach"*, Cambridge University Press, Cambridge, 1994.

Jordan D. *"C++ Object Databases: Programming with the OMG Standard"*, Addison-Wesley Longman, Inc., Massachusetts, 1997.

Keshav S. *"An Engineering Approach to Computer Networking"*, Addison-Wesley Longman, Inc., Massachusetts, 1997.

Peterson L.L. and Davie B.S. *"Computer Networks: A System Approach"*, Morgan Kaufmann Publishers, Inc., San Francisco, 1996.

Silberschatz A., Korth H.F. and Sudarshan S. *"Database System Concepts", third edition.* The McGraw-Hill Companies, Inc., Singapore 1997.

Tamer Özsu M. and Valduriez P. *"Principles of Distributed Database Systems", second edition.* Prentice-Hall, Inc., New Jersey, 1999.

Zanting D., and dr. Adriaans P.W. *"Client/server en gedistribueerde databases"*. Lanse Publishing BV, Leidschendam 1994.

"Distributed iMAN", Unigraphics Solutions Inc., Maryland Heights, 1998.

www.Cimdata.com *"Strategic Consulting for Competitive Advantage in Global Markets: PDM"*, Cimdata Inc., 2000.

www.ugsolutions.com *"iMan Information Manager"*, Unigraphics Solutions.

Internal Philips documentation:

Boland P.C. *"TPI Manual: Reasons Book"*, 1998.

Boland P.C. *"TPI Manual: Rules Book"*, 1998.

Lodewijks B. *"Unigraphs and IRIS: Way of Working"*, 1999.

Ploegmakers F. *"TPI Life Cycle: Detailed definition and implementation PPD"*, 1998.

Appendices

Appendix I: Object Modeling[1]

In object-oriented modelling, classes, objects, and their relationships are the primary modelling elements. Classes and objects model what it is in the system we are trying to describe, and the relationships between them reveals how they are structured in terms of each other. Classification has been used for thousands of years to simplify descriptions of complex systems. When using object-oriented programming to build software systems, classes and relationships become the actual code.

An object is an item we can talk about and manipulate. An object exists in the real world. It could be part of any type of system, for example, a machine, an organisation, or a business. There are objects that do not directly exist in the real world, but that can be viewed as derivatives as concluded from studying the structure and behaviour of objects in the real world. Thus objects, in one way or another, relate to our understanding of the real world.

A class is a description of an object type. All objects are instances of a class, where the class describes the properties and behaviour of one type of object. Objects can be instantiated from the class (instances created of the class type). An object relates to a class similarly to a variable relating to a type in an ordinary programming language. We use classes to discuss systems and to classify the objects we identify in the real world. Consider Darwin, who used classes to describe the human race. He combined his classes to describe his theory of evolution. The technique with inheritance between classes is also used in object orientation.

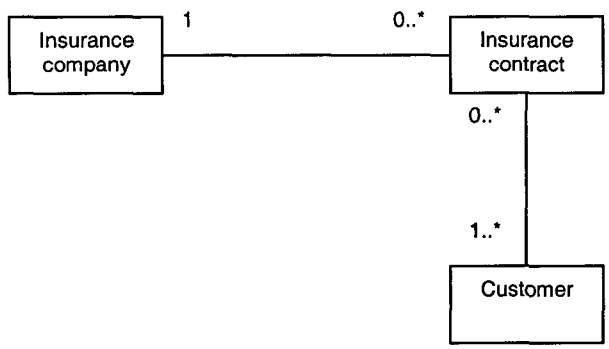


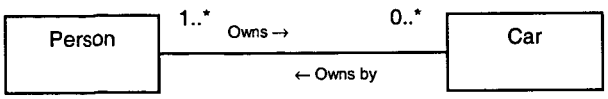
fig.1: A simple model of an insurance business. One insurance company has many (zero-to-many) insurance contracts. An insurance customer has many (zero-to-many) insurance contracts. An insurance contract is related to one insurance company. The insurance contract is related to many (many-to-one) insurance customers. The entities shown in the model are classes.

When modelling and building business systems, information systems, machines or other systems, concepts from the problem domain should be used to make the model understandable and easy to communicate. If a system is build for an insurance company, it should be based on the concept in the insurance business. A system based on the primary concepts of a business can easily be redesigned to fit new laws, strategies, rules, and so on, because we just have to adjust the differences between the old business and the new business. When machines are modelled, it is useful to mimic as closely as possible their real-world appearance, to make them easier to understand and to discuss with the customer. It is important then, that models be easy

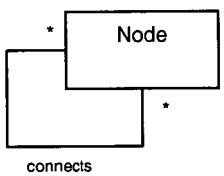
[1] Reference: Eriksson H.E. and Penker M. "UML Toolkit"

to discuss, easy to verify against functional requirements, and easy to maintain. When models are built based on how their real-world counterparts look and are based on the concepts in the problem domain, object orientation fits perfectly. The bases of object orientation are classes, objects and the relationships between them (see figure 1).

Next some simple examples of class diagrams.



A person owns many (many-to-zero) cars. A car can be owned by many (many-to-many) persons.



A network consists of many nodes connected to each other

Bibliography: Eriksson H.E. and Penker M. *"UML Toolkit"*, New York, Wiley, 1998.

Appendix II: TPI Life Cycle

The following description of the TPI Life Cycle is extracted from the internal Philips document "TPI Life Cycle", author Frank Ploegmakers.

TPI (Technical Product and Process Information) consists of all technical information that is needed to develop and manufacture a specific product.

Terminology: Organisation PPD

- Designer.

The one originally creating the design and therefore responsible for the technical correctness of the original content. This creator-ship keep unchangeable identified on documentation even though the designer could resign, retire or move. Due to the technical knowledge towards the design this person also participates in judging changes of the design. The designer can also, but not necessarily, be the Owner.

- Owner.

The one presently responsible for the whole "design system", the "why's and what" of the design system. The Owner needs to be consulted regarding the technical content until ownership is changed. The Owner is judging changes as per today and is uniquely identified in the TPI system IRIS. The Owner could be the designer if applicable.

- (Segment) Project Manager.

Organises and manages on IPC, PPD and Segment level the Development Project regarding the creation of TPI, technical reviews and has the responsibility for transferring the TPI at an agreed milestone. Part of the project is the TPI Checklist.

- Project Information Assistant.

Administer of the documentation of the product structure, the availability and administration of TPI and supplies overall support within a Development Project towards TPI and IRIS.

- Segment Manager:

Responsible for TPI conform the design rules. Decides who should do the work within the segments but can delegate this to the Group Manager. Appoints Owners for all segments owned TPI e.g. all TPI have always an Owner, which is available in the PPD organisation.

- Group Manager:

Organises, sets conditions and could decide who should do the work on behalf of the segment manager. Responsible for the technical integrity and consistency of the TPI.

- Local Development Manager:

Responsible for local TPI. Decides who should do the work within the local development organisation. Local Development Manager appoints Owners for all local owned TPI e.g. all TPI has always an Owner which is available in the local organisation. At an agreed milestone in a development project the Local Development Manager takes up the authority and responsibility, to (1) assure that the product is conform specification e.g. TPI = Production and (2) keeps the TPI up to date.

Terminology: TPI System IRIS

- TPI Creator.

A TPI Creator is a user of the TPI system IRIS and creates all documents. A TPI Creator may decide to release data to a higher status (provisional, consolidated, final or cancelled) and therefore may initiate a Release Procedure. Before the actual release of information takes place,

users with the role of approver must approve the release. The initiator of the release can assign users (Co-Designers, Owners or (Segment) Project Managers) to be approver. Reasons for being an approver could be co-engineering, technical knowledge, project management or final inspection. After data has been released to a higher status like Final or Consolidated a TPI creator is no longer able to apply any changes to the data. To change data however he can, like any other user, initiate a Change Procedure. The Change Proposal has to be approved by users with the role of approver, such as Owners.

- Owner.

The Owner is someone who is identified within IRIS as being the Owner of that specific TPI on system level. Owner is stored as meta-data within the system.

- Responsible Person.

The system gives the possibility to indicate the responsible person for: (1) products (12nc) and (2) TPI documentation stored within the system. Responsible person is stored as an attribute on item (product) or document level within the system.

- Approver

An approver is a system role regarding the release and change of TPI. During these procedures approvers give one's approval with respect to the concerned released or changed data.

The ***TPI Life Cycle*** is as follows defined:

Step in the TPI Lifecycle		
	Development Phase	*Explanation of the step*
1	Appoint the "designer"	Segment Manager or the properly addressed person (Group Manager) decides who should do the work. (Segment)Project Manager is informed.
2	Designer does the design work including the necessary creation of documentation	Designer is the TPI Creator – until formally decided otherwise. In this case, the "designer" may be doing product or process design and development work, and creating the necessary TPI. Segment (group) Manager and Local Development Manager are the Responsible Person.
3	Technical review using Release Procedure of TPI in IRIS	Project Manager makes technical review happen in accordance with Distributed Development Policy and according to the local used and agreed TPI checklist. Technical review with respect to all TPI from Type and Technology projects. At this point, the Segment Manager *may* decide to change Ownership, if appropriate.
4	Transfer of TPI at an agreed milestone	The stage at which transfer is executed is agreed between the Segment Manager and the Local Development Manager. (As a guideline, normally, this stage will be Design Release / Transfer Release for products and Industrial Feasibility for technology projects). It is the responsibility of the Project Manager to ensure that this happens. This does not necessarily involve a change of Ownership at this stage. The PPD stays being Owner of the TPI, however the Local Development Manager or the properly addressed person has the authority and responsibility for keeping the TPI up to date (TPI = Production).

	Manufacturing Phase	*Explanation of the step*
5	Change proposals (CP) for TPI data beyond Step 4	Anyone can generate a change proposal (CP), and reviews it with the Owner. In order to manage this process-dedicated information / communication lists have to be made for each component and/or process. These lists have to contain the people to be involved in case of a change. The responsibility for these communication lists has to be defined. If the current Owner is in agreement, the change proposal follows standard procedure - the changes needed are done in the TPI through the one initiating the proposal and implemented into current practice as appropriate. There are no specific consequences to the ownership of the TPI.
6	Disputed changes	If the Owner does not agree with the proposed change, the Local Development Manager must discuss with Segment Manager.
7	Dispute outcome 1 - Withdraw proposal -	Local Development Manager (or Local Designer) withdraws change proposal.
8	Dispute outcome 2 - New derivative or variant -	New TPI is created (derivative or variant) with new Ownership by Local Development Manager, the Segment Manager and current Owner are informed. Local Development Manager is responsible for getting the changed data into TPI. This would apply to, and/or create, 'derivative' or 'variant' products / components or processes. A new 12nc with appropriate Owner and TPI has to be created. It is assumed that all such changes remain within the limits set by the overall design rules.
9	Dispute outcome 3 - Local development managers do not agree -	The outcome should be decided by discussion across all affected sites. The Local Development Manager in the proposing organisation should link and co-ordinate the discussion, and the PPD Segment Manager and Owner of the TPI should be kept informed throughout the process.

	Cancellation Phase	*Explanation of the step*
10	Cancellation of TPI	The Segment Manager or Local Development Manager initiates the cancellation of the relevant TPI. A proposal for cancellation is treated as a change proposal. Cancellation of components or processes can be combined with the cancellation of related TPI. Cancellation of a component can only be done when all relevant parties are involved (Product Management, PPD, IP(S)C's, Purchase, etc. This will generally move the status of the data from "final" to "cancelled". The meta-data will remain accessible in the system, and the volume data [actual data file] will be archived in accordance with agreed practice.

Lifecycle independent activities.

Lifecycle independent activities		*Explanation of the activity*
1	Establish Design Rules	Segment Manager is responsible for TPI conform the design rules. These are part of the TPI, and the Segment Manager is the Owner of the rules. The Segment Manager could delegate this ownership to Group Managers.
2	Change of ownership - at any stage in the life-cycle -	Any changes of Owner should be authorised by Segment Manager or Local Development Manager(s). All TPI has in any stage, except after cancellation, an Owner.
3	TPI life-cycle during starting up or transfer of production between sites	As production is started up /or transferred to a new site, that site will have access to all concerned data - both original 'baseline' design and also variants. Data for the "receiving" site will be based on "latest" versions applicable in current manufacturing - unless local production conditions or equipment make it inappropriate. In practice parts of the TCA (Product Structure) can be owned at different places. A shared product structure is possible, however each component or process TPI has a unique Owner within the BG.

Appendix III: Release & Change Procedures

Quality level
In IRIS, objects can be released to a certain Quality Level. The Quality Level is used to indicate the value of the information concerned. "Value of information" is defined as the types of decisions that can be and are allowed to be made on the basis of information (e.g. *provision* information is not to be used to order equipment). Instead of "Quality Level", the term "status" of a document is sometimes used. Quality levels are as follows defined:

Document Quality Level	**Value of Use**	**Users**
Private	Document contains information not to be used before the designer (group) is consulted.	1 or few
Provisional	Document can be used for estimates; prices, preliminary tooling	project team
Consolidated	Document is accurate and complete enough to execute definitive price estimations order equipment and make models. During changes the consequences for equipment and component-suppliers must be taken into account	Whole BGDC
Final	Document is free for use in production. During changes the consequences for all involved organisations have to be taken into account.	Whole BGDC
Cancelled	Document and information is not valid anymore	Whole BGDC

BGDC: Business Group Display Components

Datasets and Item-revisions are subject to a certain Quality level. When a new object is created (or imported) in IRIS, it's Quality Level is considered to be "Private". IRIS does not have an indication for this QL. If the "Release status" column for an Object is blank, no Release Procedure is yet executed and, therefore, the QL is considered to be "private". Using a Release Procedure carries out changes of the QL to a higher level.

Release Procedure
The Quality Level of an object in IRIS is changed using an electronic release procedure. This procedure involves at least two IRIS users: someone who proposes the release and an approver (who - electronically - agrees with the release). There can be more than one approver, however. The IRIS system doesn't prescribe who should be involved in the different release procedures. The role of approver can be assigned to any user that is available in the drop down list in IRIS. There should a clear understand that it's organisational demands that governs assignment of approvers to specific release procedures.

By default an object is set to Quality Level *Private*. This Quality Level is not visual in IRIS, but any object without a QL is considered *Private*. There are 4 Quality Level Releases:

- Release to provisional
- Release to consolidated
- Release to final
- Release to cancelled

The 4 available Quality Level Releases are each dived into 3 different procedures. The number in the procedure name reflects the number of approvers that will have to be assigned in that procedure. E.g. using the procedure "Release to Consolidated 3" will imply assigning 3 IRIS users as approvers to the release. The actual release will only take place when all 3 approvers

have agreed on the proposed release. Using the procedure "Release to Final 5" means the QL will only change when all 5 approvers that have been assigned, agree on the release.

The Quality Level of an object increases with the value of the information. As information becomes more "mature", in time more people in the organisation will come to depend upon it. As a consequence, more people or parts of the organisation will be affected if the information is to be changed. It is therefore vital to ensure that:

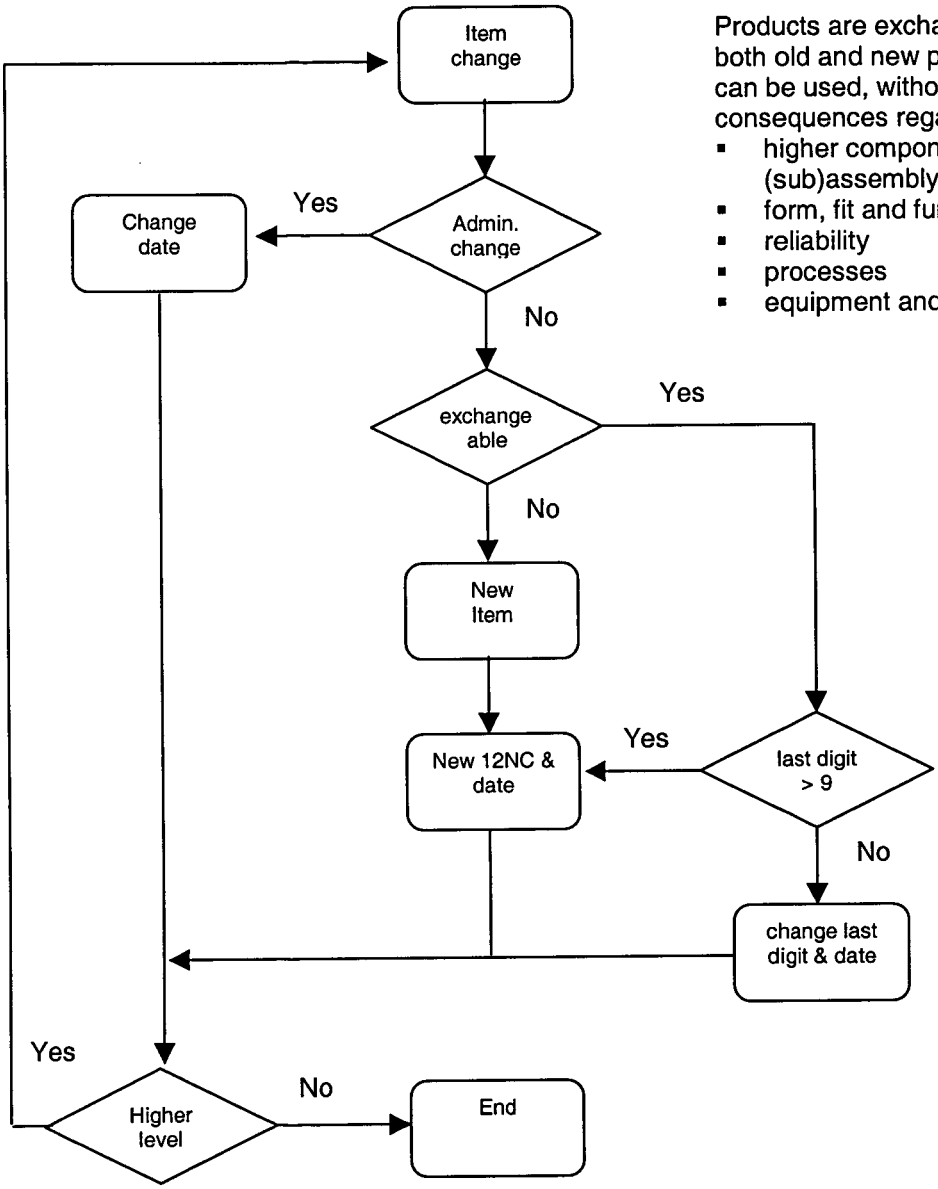
- All relevant people are involved in the decision to change (which is ensured through the change procedure).
- All released data is protected against accidental changes, by resetting the access rights in a specific way. This resetting of access rights depends on the Quality Level of the information. If the Quality Level is "high", the information should relatively be better protected.

remark:
Items that are part of a project may have a certain Milestone. Milestones will not be indicated in the "Release status" column. Currently, IRIS will not support automatic Milestone indications. Milestones are to be included in a document (Survey of Milestones) manually.

Reference: TPI Manual, Reasons Book (Philips internal documentation)
TPI Manual, Rules Book (Philips internal documentation)

Appendix IV: Item Revision Rule

In order to decide whether a new item or item-revision has to be created, the following Revision Rule[1] is used.



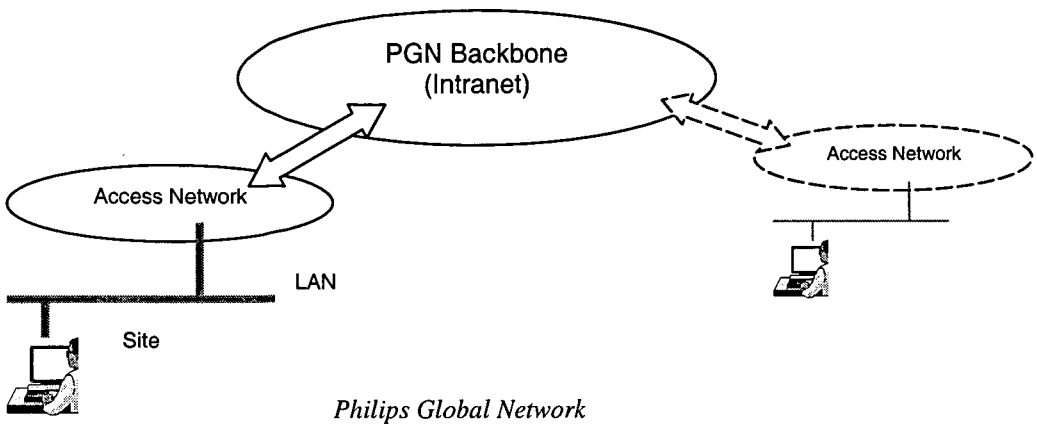
Products are exchangeable if both old and new products can be used, without any consequences regarding:

- higher component / (sub)assembly
- form, fit and function
- reliability
- processes
- equipment and tools

[1] references: TPI Manual: Rules Book (internal document)

Appendix V: The Global Network

The Philips Global Network (PGN)[1] is divided into a Global Backbone Infrastructure and the Access Networks per country. The sites are connected through Access Networks to the Global Network.



Philips Global Network

Bandwidth:

Backbone (from Eindhoven)	
to Ger:	10 Mbps
to Aut:	2 Mbps
to Esp:	256 Kbps
to Chi:	1.5 Mbps
to Tai:	1.5 Mbps
to Bra:	512 Kbps
to USA:	3.5 Kbps
to Fra:	2 Mbps
to Gbr:	3 Mbps

Access Networks (sites)	
Ottawa:	3 Mbps
Ann Arbor:	512 Kbps
Capuava:	1 Mbps
Hsinchu:	1.5 Mbps
Chupei:	1.5 Mbps
Chungli:	1.5Mbps
Nanjing:	256 Kbps
Barcelona:	256 Kbps
Hamburg:	2 Mbps
Eindhoven:	10 Mbps
Sittard:	1 Mbps
Durham:	512 Kbps
Blackburn:	192 Kbps
Washington:	384 Kbps
Simonstone:	384 Kbps
Dreux:	768 Kbps

Round Trip Times:

within Europe:	< 170 msec	between Europe – North America:	< 420 msec
within North America:	< 300 msec	between Europe – Asia:	< 720 msec
within Asia:	< 400 msec	between Europe – S. America:	< 500 msec
within South America:	N/A.	between N. America – Asia:	< 720 msec
		between N. America – S. America:	N/A.
		between S. America – Asia:	< 720 msec

[1] Reference TPI-Infrastructure, Network Communications: PGN (TVB-931-00-HD-D0018) dd. 21-01-2000