INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.352

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(03/93)

MAN-MACHINE LANGUAGE

DATA ORIENTED HUMAN-MACHINE INTERFACE SPECIFICATION TECHNIQUE – SCOPE, APPROACH AND REFERENCE MODEL

ITU-T Recommendation Z.352

(Previously "CCITT Recommendation")

FOREWORD

The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the International Telecommunication Union. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, established the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

ITU-T Recommendation Z.352 was prepared by the ITU-T Study Group I (1988-1993) and was approved by the WTSC (Helsinki, March 1-12, 1993).

NOTES

1 As a consequence of a reform process within the International Telecommunication Union (ITU), the CCITT ceased to exist as of 28 February 1993. In its place, the ITU Telecommunication Standardization Sector (ITU-T) was created as of 1 March 1993. Similarly, in this reform process, the CCIR and the IFRB have been replaced by the Radiocommunication Sector.

In order not to delay publication of this Recommendation, no change has been made in the text to references containing the acronyms "CCITT, CCIR or IFRB" or their associated entities such as Plenary Assembly, Secretariat, etc. Future editions of this Recommendation will contain the proper terminology related to the new ITU structure.

2 In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

© ITU 1994

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the ITU.

CONTENTS

Page

1 Scope of specification technique 1

2 Approach 1

2.1 Data versus functions 1

2.2 Consequences 1

2.3 Data design 3

3 Reference model 3

3.1 Overview 3

3.2 Scope 4

3.3 Subdivision of the external layer 6

Annex A – Guidelines for HMI developers 6

A.1 Introduction 6

A.2 Method 7

A.3 Data design 10

Appendix I – Additional guidelines for data designs 12

I.1 General 12

I.2 Design of classes 12

I.3 Design of identifiers 16

I.4 Derived data and level of detail 21

SUMMARY

This Recommendation provides a framework for specifying harmonized HMIs.

The Recommendation points out the importance of identifying the right application area and the great impact this choice will have on the final design of the HMI. Therefore, a data oriented approach, rather than a functional approach, is proposed.

A three-layered reference model for HMI is introduced and explained. This reference model allows the centralization of data definitions for HMI and derivation of external presentations in a harmonized manner.

Annex A provides guidelines for how to carry out development of HMIs in accordance with the chosen reference model and formalism. The development method is not a waterfall-type development process. This annex also provides a framework for design of data seen at the HMI.

Appendix I provides additional guidelines for data design.

The mapping between the HMI reference model and the TMN functional architecture is under study.

Areas that this Recommendation has relations with and impact on, include:

- mapping to TMN specifications,
- software standardization.

DATA ORIENTED HUMAN-MACHINE INTERFACE SPECIFICATION TECHNIQUE – SCOPE, APPROACH AND REFERENCE MODEL

(Helsinki, 1993)

1 Scope of specification technique

The objective of this specification technique is to specify the data seen at the HMI and the grammar for these data. It is not the objective to define how the telecommunication networks operate internally.

The specification of the data is used by the end users prior to the implementation:

- to assess what an application is about;
- to ensure that its grammar is correctly understood;
- to ensure that its terminology is well chosen; and
- as a central part of a contract for the implementations.

The specifications should also be applicable as a means for:

- user guidance to the permissible and/or mandatory structure of the data and the explanations of the data;
- accessing the "instances" of the data for that application.

From this it follows that the specifications should be readable by and applicable to both designers and end users.

The designers can be both HMI designers and software designers. The end users can be both OA&M operators inside the Administrations and external customers administering their own services.

2 Approach

2.1 Data versus functions

Functional specifications identify and decompose the tasks being performed in different functional areas. This decomposition of activities does not separate the objects that the functions apply to and how these objects are manipulated.

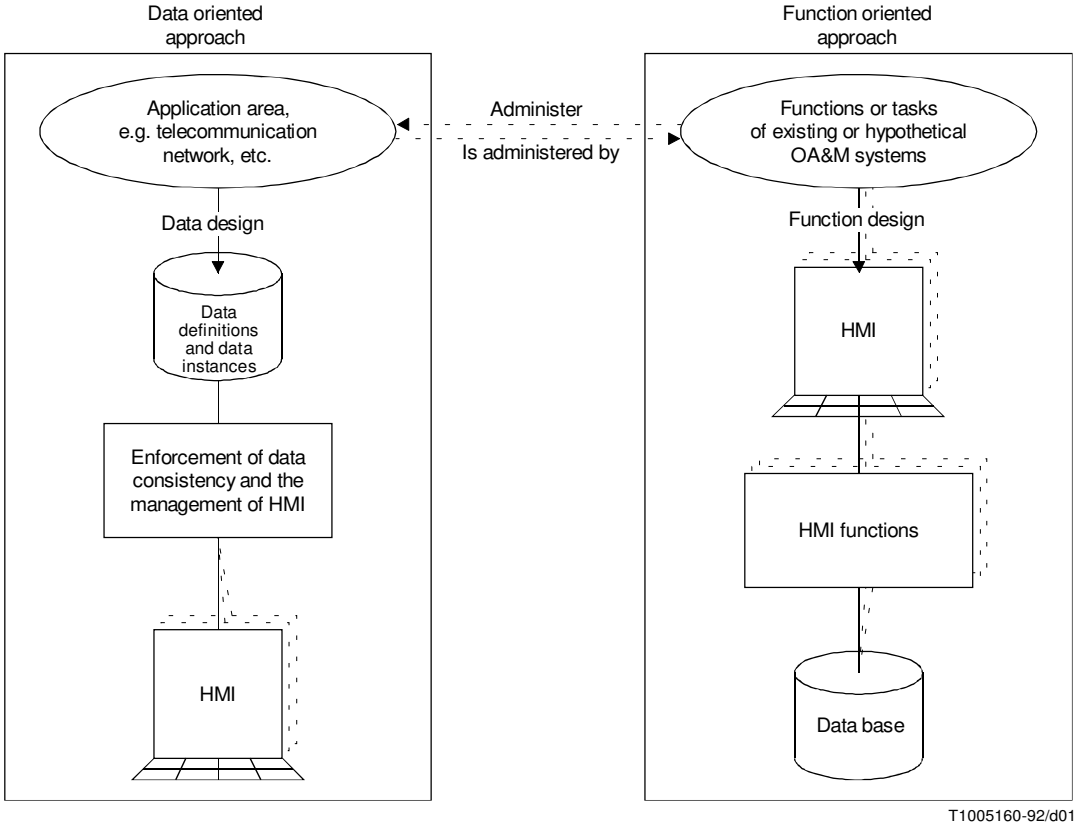
Data oriented approaches focus on identifying the data to which the functions apply, independently of the functions themselves. The means of manipulation on the data can most often be defined in a generic way – independently of the different types of data which are identified. It is very important to recognise that the two approaches tend to study two different universes of discourses (see Figure 1). Function oriented approaches study and decompose the tasks going to be performed in an organization. Data oriented approaches study and decompose the application area being administered by the organization. As a result of the data oriented approach a more coherent view of data is likely to be available for presentation to the human user.

2.2 Consequences

The implications of choosing one out of the two approaches are profound. The functions or tasks of any organization can change more rapidly than the data manipulated by these functions or tasks.

The function oriented approaches tend to analyse an old or hypothetical information system. This unavoidably conserves both existing characteristics and the designers' preconceptions into the design of the future system.

This observation has several further implications.



NOTES

1 In the function oriented approach the analysts study and decompose the tasks performed by the organization. For each task the human-machine interfaces are identified and specified. This is illustrated by the arrow labelled "function design" in the right hand part of the figure. The final function oriented system is implemented by developing one process for each individual HMI function, using one or several common data bases.

2 In the data oriented approach the analysts study and design common data definitions for the entire information system. This is illustrated by the arrow labelled "data design" in the left hand part of the figure. The processes that enforce consistency of the data can be made common for all data. The HMI does not have to be separated for each task, but can be common for several tasks. Note that the two approaches tend to study two different universes of discourse, as depicted by the two ovals in the upper part of the figure.

FIGURE 1/Z.352

Alternative approaches to HMI development

In the function oriented approaches the sequences of tasks to be performed are frozen into the implemented system; while the data oriented approaches provide more flexibility for changes in the sequences of tasks.

Both training and usage of systems designed according to the two different approaches tend to be very different. In the function oriented designs the users learn and use the different functions of the system and learn only implicitly about the universe of discourse, i.e. application area, and the structure of the data being manipulated by the functions. In the data oriented approaches the emphasis is put on understanding the application area being administered by the system and defined by the structure of the data of the system; the training on tasks to be performed is given less priority.

While function analysis can help to identify needs for data, function analysis does not help to identify data for additional functions, for new users or to identify unrelated data. Therefore, function analysis can constrain rather than support innovation and the questioning of existing data definitions. While the decomposition of functions can be an aid to the identification of needs for data, this decomposition is not of relevance in the final documentation of the data. Nevertheless, it may be useful to record the functional needs and motivation which led to the identification of data along with the definition of those data.

In the function oriented approaches the data are identified at the lowest level of decomposition of the functions. The identified data can be inputs to or outputs from the different functions. The identification of data separately in each function does not address and question the overall structure, design and definition of the data. Hence, the data definitions can continue to exist unharmonized and defined in such a way that they are unknowingly inefficient in use to the end users.

In the function oriented approaches separate design phases address the grouping of data into logical file structures, appropriate for the functions to be performed. These design phases take into account the access to and manipulation of data, but are usually not capable of reconsidering the overall design of data for that application.

2.3 Data design

In the data oriented approaches the data are identified independently of functions. The analysts and designers:

- observe the application area being administered by the organization;
- design data definitions for that application area.

The primary objective of the data oriented approaches is to design efficient data definitions for the chosen application area. For the design of data, a deep knowledge of the application area being administered is required and must be acquired.

The data definitions are important, because they can help the organization understand and perceive its functions and how it can perform its tasks.

3 Reference model

3.1 Overview

The data oriented approach allows for partitioning all HMI specifications and all software into a layered architecture:

- The external layer handles the presentation and manipulation of data. It also handles mapping to and subsetting of the application data.
- The application layer is that layer of the HMI reference model which is concerned with the definition of data and their behaviour.
- The internal layer is outside the scope of HMI. The layer is supposed to take care of the internal storage, accessing, implementation and communication of data and their behaviour.

Each layer of the layered architecture is partitioned into schemata, processes and populations. Data of each layer are mapped to data of adjacent layers only. A schema contains the data definitions, including constraints and derivation rules for the corresponding population data. A population contains the data instances which are enforced by a process according to the rules expressed in a corresponding schema. Only some aspects of the populations [HMI(s) and Data base(s)] are depicted in Figure 2. A process implements the enforcement of the rules found in a schema on the data instances in a corresponding population.

The application layer is a centralized resource of application data and their behaviour. The specification of the common terminology and grammar of all HMI data is not dispersed into several external functions, but is non-redundantly defined in the application schema.

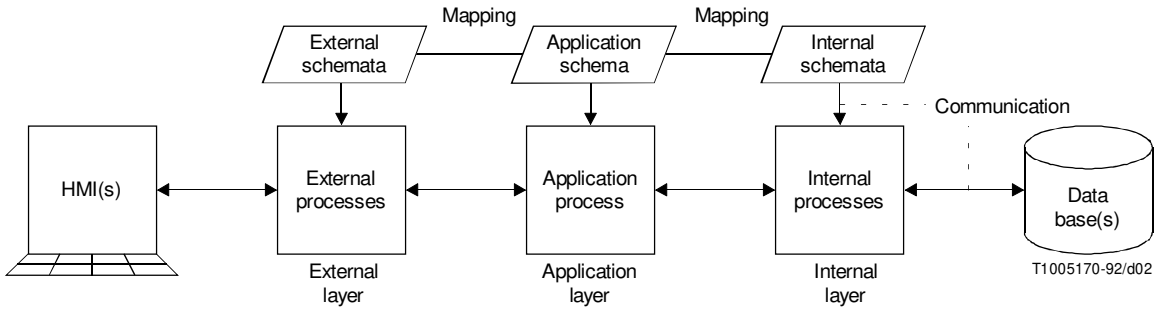


FIGURE 2/Z.352

The three-layered architecture

3.2 Scope

The scope of the reference model of the HMI work is illustrated in Figure 3. The three-layered architecture is used only to define and illustrate the scope of the HMI and is not intended to impose restriction on the implementation.

The three layers are:

- the HMI external layer;
- the HMI application layer;
- the HMI internal layer.

The HMI application layer is controlled by one application schema, being a centralized resource, containing the specification of the structure and the dynamic behaviour of all HMI data.

The HMI external layer can contain several external schemata. The external schemata specify the detailed concrete syntaxes (layout) at the HMI and the access rights for each presentation.

The mappings between the application schema and the external schemata state the chosen subsetting and derivation for each presentation.

The internal layer is outside the scope of HMI. The layer is supposed to take care of the internal storage, accessing, implementation and communication of data and their behaviour.

The end users primarily use (the presentation of) HMI population data (instances), according to the external schema specifications.

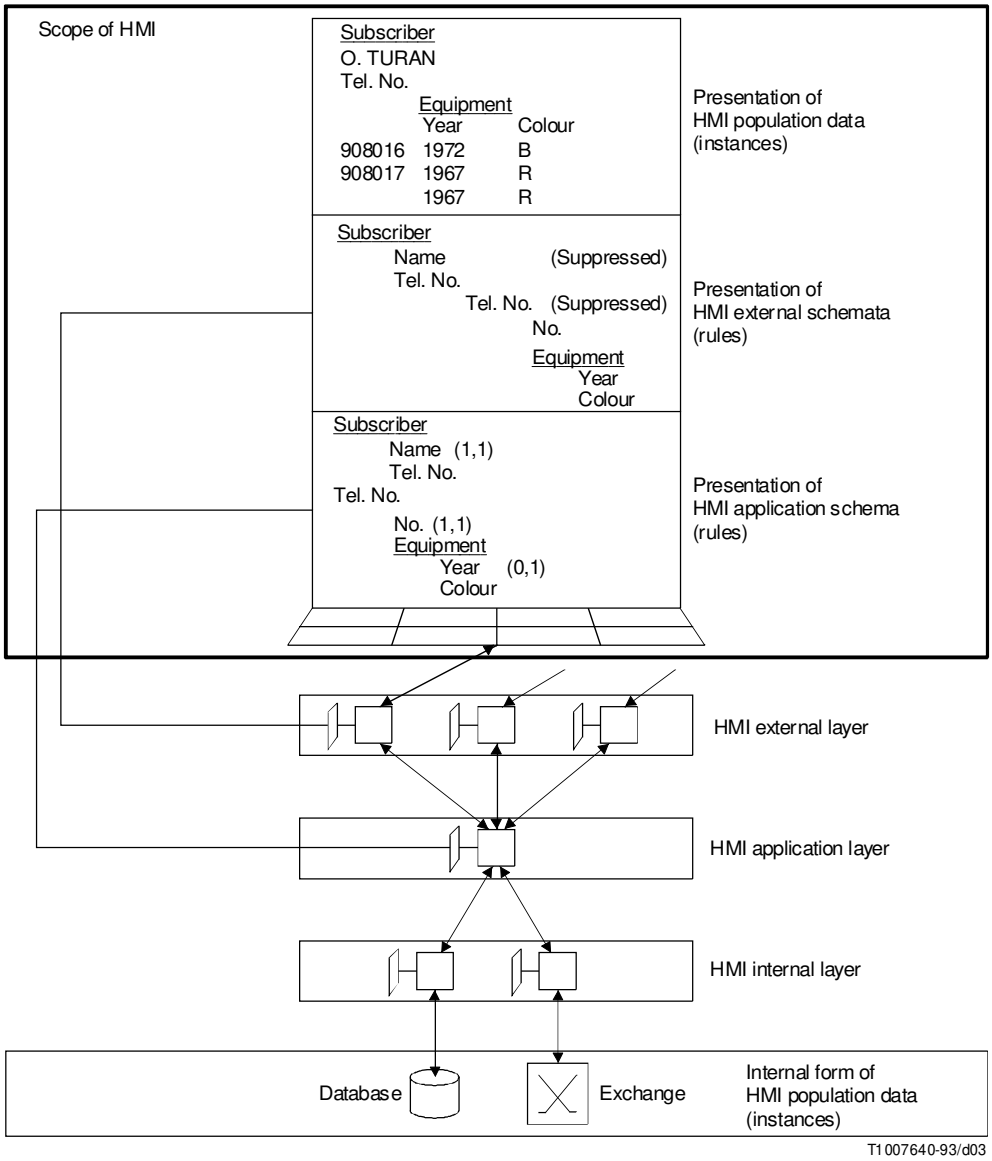
In order to know which presentations are available, the end users need access to the external schemata and their contents.

However, the end users also need access to the definition of the data, their behaviour and interrelations. This information is provided from the application schema.

The end users do not normally need access to the internal aspects of the systems. The scope of the HMI work is therefore defined as consisting of the presentation and manipulation of:

- the HMI population data;
- the HMI external schemata;
- the HMI application schema.

The end users' requirements on the syntax and behaviour of the presentation of the HMI population data give requirements for the (presentation of the) external schemata. The (presentation of the) external schemata give requirements for the (presentation of the) application schema, which must be capable of specifying (the union of all) HMIs.



NOTES

1 The scope of HMI is illustrated inside the upper rectangle. In addition to the population data, the scope of HMI is shown to include the presentation of both the specifications in the external schemata (for the presentation of the HMI population data) and the common specifications in the application schema (for the structure and behaviour of the HMI population data).

2 The internal form and storage of data are shown in the bottom rectangle. The end users do not need to access to the internal form of the data.

3 The screen layouts and partitioning into windows are examples, illustrating the scope of HMI only, and are not intended to illustrate existing or to impose future recommendations. The screen can contain graphics as well, but that is not illustrated. Also, the database and the exchange are examples only.

4 Within the rectangles depicting the layers the small parallelograms represent schemata and the small squares represent processes. The two-way arrows represent population data flowing both ways.

FIGURE 3/Z.352

Scope of reference model

3.3 Subdivision of the external layer

The external schema is concerned with the specification of data at the HMI for a specific presentation. A subdivision of the external schema is provided as follows (see Figure 4):

- the contents schema specifies the structure of the selected data and their relationships for a specific presentation;
- the layout schema specifies the way in which the data are to be presented to the human user.

The contents of the schemata and the mappings between the schemata provide specifications from the human (designer's and user's) point of view.

These contents and mappings state nothing about the final implementation of the system. For example, all the specifications can be be compiled into one single functional block.

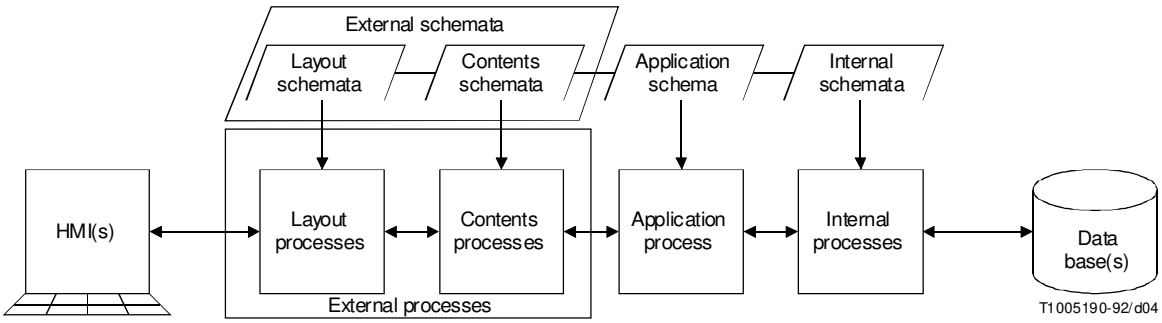


FIGURE 4/Z.352

Detailing of the external schemata

Annex A

Guidelines for HMI developers

(This annex forms an integral part of this Recommendation)

A.1 Introduction

This annex contains:

- subclause A.2: Method;
- subclause A.3: Data design.

A.2 Method

A.2.1 Purpose

The purpose of the method is to:

- identify the scope of human-machine interface specifications;
- identify activities which are strictly needed in order to develop human-machine interfaces in accordance with the reference model of this Recommendation;
- identify necessary sequences of these activities;
- give some guidelines for good data design.

For information on the scope see clause 1. For guidelines on data design see A.3.

A.2.2 Overall development process

This subclause outlines the overall development process for human-machine interfaces. The overall development process is implied by the three-layered architecture of the reference model, but is not influenced by the chosen formalism of the human-machine interface specification technique.

The overall development process can be divided into the following activities:

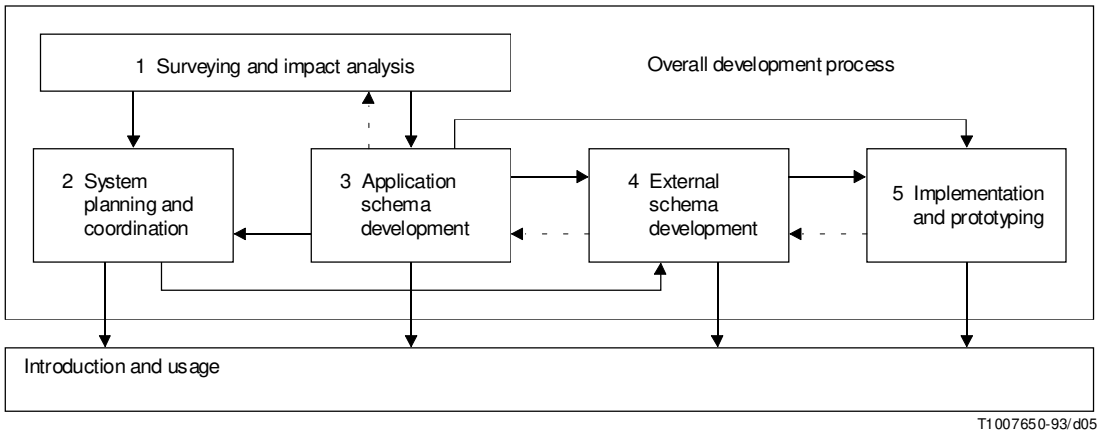
1) surveying and impact analysis;
2) system planning and coordination;
3) application schema development;
4) external schema development;
5) implementation and prototyping.

Introduction and usage of specifications and implementations according to the specifications are considered to be outside the scope of the overall development process.

A brief introduction of these activities is as follows:

- Activity 1, Surveying and impact analysis, is not the subject of this series of Recommendations. It can include:
  - analysis of the application area;
  - analysis of related systems, e.g. involved organizations and tasks;
  - development of sketchy designs;
  - stating objectives of the new system(s);
  - analysis of economic, organizational, personnel and other consequences;
  - defining scope, limits and introduction of the future system(s).
- Activity 2, System planning and coordination, is not the subject of this series of Recommendations. It can include:
  - development of proposed organization of work procedures in the user community.
- Activity 3, Application schema development, is the main focus of the data oriented HMI specification technique. The proposed development process for the application schema is described in A.2.3.
- Activity 4, External schema development, is currently left for further study. It will include:
  - specification of the contents of views (contents schemata) of the application schema;
  - specification of concrete HMIs (layout schemata).

– Activity 5, Implementation and Prototyping, is not the subject of this series of Recommendations. It can include
  – HMI application software development;
  – verification and validation of the developed HMI.



NOTE – Solid arrows indicate the main data flow. Dashed arrows indicate feedback. Activities 1, 2 and 5 are not the subject of this series Recommendations.

FIGURE A.1/Z.352

Overall development process

A.2.3 Application schema development

An important consideration in the application schema development is the identification of the right application area. For discussion of this see 2.

In a data oriented approach the analysts and designers should start by observing and modelling the fundamental entities which exist within the application area or domain to be managed. Definitions based on these entities are likely to remain stable, since they themselves do not change as rapidly as tasks and routines managing data about the entities.

Another important consideration is to identify and understand the perspective from which the application area is being defined, as different perspectives will lead to different application schemata. This is exemplified by the network traffic management application area. Traffic management can be viewed from the network perspective (an administrator looking at the traffic between exchanges) or from the exchange perspective (an administrator looking at the traffic being handled by a single exchange and viewing circuits to other exchanges from the perspective of that exchange).

In the development of an application schema for a large and complex application area (domain), it is inevitable that the application area has to be divided into sub-areas as it is not feasible to develop one all-embracing application schema. The manner in which this division of the application area is carried out is likely to affect the effectiveness of the development process. It is suggested that the division should be carried out in a manner which aims to reduce duplication of data between application schemata for different sub-areas. The process of dividing the application area into sub-areas must be seen as an iterative process. It is likely to be affected by the results of other later activities.

Activity 2, Application schema development, can be organized in the sub-activities shown below and illustrated in Figure A.2:

2.1 development of object classes and references;

2.2 development of attribute classes;

2.3 development of behaviour;

2.4 data coordination.

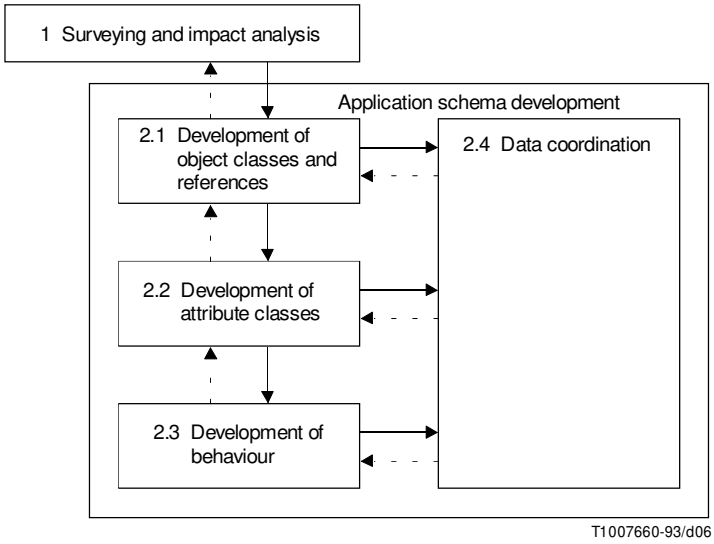


FIGURE A.2/Z.352

Application schema development

The objective of application schema development is to define the object classes, references (relations), attribute classes and behaviour. The design method consists of identifying them in the given order, but accepts that the process may require several iterations. These iterations allow a step-by-step refinement process. For example, data items which are initially perceived to be attributes may be perceived as objects after further investigation.

Activity 2.1, Development of object classes and references: the goal of this activity is to analyse a particular application area to identify the object classes to be managed from an HMI perspective and the references between these classes. Object classes may contain object classes in a hierarchy. Graphical documentation of the Application Schema is a valuable tool for this activity.

The process of identifying object classes and their references involves observation of object instances and their relationships. Classes are created by a process of generalization, involving the identification of instances with common characteristics. The designer has to make decisions as to the degree of generalization which should be applied.

Identifier attributes should be considered within this activity. See Appendix I.

Sometimes an application schema needs only to contain specifications for a single moment of time. Often, however, an Application Schema may also need to contain specifications related to past and/or future states. In this case, when the time-independent terminology has been decided, the handling of past history or future planned events may be treated as an extension to the time-independent data definitions. Alternatively, the handling of time may be included in the earliest data designs. Addition of time-dependent features to the data definitions is likely to have an impact on the definition of constraints which is carried out in activity 2.3.

Activity 2.2, Development of attribute classes: when the object classes have been identified, the following properties will be defined:

- attribute classes, defining the inherent characteristics of an object class, e.g. the operating characteristics controlling the way the resource will operate, the ability of a resource to provide a service to an end user;
- attributes classes which contain subordinate attribute classes, e.g. a subscriber's name consisting of parts;
- permissible value classes and ranges of value classes of attributes;
- permissible syntaxes for complex value classes of attributes.

Activity 2.3, Development of behaviour: in this activity the following are defined:

- constraints on the values and other data;
- derivation rules for new data from existing data.

This can be achieved using the function construct of the formalism.

Activity 2.4, Data coordination: data coordination is left for further study. It can include:

- comparison of data definitions from different subareas; identification and resolution of overlaps and inconsistencies;
- updating of a common dictionary.

A.3 Data design

A.3.1 Introduction

This subclause provides a framework, and additional guidelines for data design are found in Appendix I.

A.3.2 Framework

It should be emphasized that data design work is often "first-of-a-kind", unlike implementation which more often can be routine work. Hence, the amount of work needed to reach a certain quality of data design can be rather unpredictable. The development process is not a mechanical transformation of fixed and settled requirements, but is an investigation into an open environment. Success is dependent on having good knowledge of the application area, being able to foresee the impact of alternative choices and being able to revise, generalize and systematize diverse phenomena in the application area.

The HMI specification technique specifies the required common form of all specifications, but provides no substance for the particular application area. The data designs can obey the required form, but still be bad designs from the point of view of the end users.

The designers of application schemata/data definitions for HMIs must have in mind:

a) the allowed and implied instances of the definitions;

b) the allowed and implied HMI presentations of these instances.

The defined data classes and instances can describe entities in an application area. Therefore, the designers have to observe and investigate this application area when designing the data. The identification of the right application area is no trivial task, e.g. you can easily confuse modelling the old information system rather than modelling its application area.

The data instances and classes will be used by some users. Therefore, the designers have to observe and investigate the needs of these users when designing the data. The identification of the real users is no trivial task, e.g. the terminal operators may only be communicators of information to the final users.

In addition to the investigation of:

- user needs;
- appropriate application area;

the designers will have:

- knowledge of the formalism for the application schema;
- guidelines for good data designs, satisfying a) and b).

The last item will also include knowledge about treatment of time, coordination of applications, reference models for information systems, etc.

The users' needs will provide the (informal) guidance to what part of the application area shall be described and administered. The selection of phenomena in the application area and the understanding of the application area is dependent on the language constructs used by the observer when registering his observations. The finally selected phenomena described in the application layer are dependent on the formalism used for this description. The language constructs for the original observation and the final description may be one and the same.

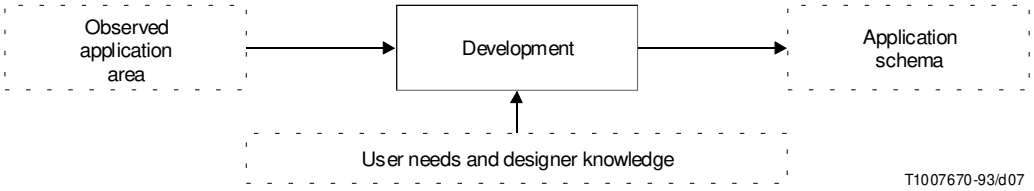


FIGURE A.3/Z.352

Development of HMI data

Even though users' needs are taken as input to the development, the development is considered to be a function (many-to-one mapping) from the observed application area to the final application schema. Hence, there are no alternative descriptions of the same fact. This statement disregards the permission to add alternative synonymous names in step 7-10 below.

The character of the development function is dependent on the formalism used for observing the application area and writing the application schema. Unfortunately, the function may also be dependent on choices made in the development method used and on the data definitions already contained in the application schema.

One way to perform the mapping from the observed application area to the application schema can be:

1) select example entity instances from the application area, including their references;
2) design application schema proposals for each instance example;
3) study similarities between the schema proposals and try to unify them;
4) redefine the examples according to new schema proposals;
5) create an integrated schema for all proposals and finalize the iteration;
6) select the relevant portion of the overall schema to be carried further;
7) assign suitable end user names to the the final schema (objects and references);
8) add and design identifier attributes;
9) add and design other attributes;
10) add and design values, constraints and derivation rules.

The last step will include the definition of derived data, providing overview of other data.

Note that steps 1) and 6) are pure selection functions (one-to-one mappings) and are not many-to-one "abstractions"/ "generalization". If we allow some instances to be schemata (i.e. prototypes) for other instances, then step 2) can be considered empty or as a one-to-one mapping as well.

The reason for considering the development function from the application area as basically being a selection process is that we want to stick to an isomorphic mapping between a description and its described world. If not, we have to know the entire development function in order to interpret correctly the data against the claimed described world; having the data only, we would have no certain knowledge of the structure of the described world.

In some applications there is no described application area, e.g. in an electronic mail system the data may constitute the real world itself. The treatment of time can be integrated in the above steps or added as a separate step.

While most data definitions are defining data about an application area of the information system, e.g. about the telecommunication network, there is nothing prohibiting the description mapping from being nested. Therefore, the information system itself can be described inside the EDP system. Hence, we can describe information handling processes, information flow, control flow, etc. Some of the described processes can be automatic, others can be manual.

From the previous paragraph we recognise that to do information flow analysis is in principle nothing different from designing data definitions about the application area of the information system. To define information flow is just to describe another application area. The two application areas are related. And the two application areas can be defined by using the same kind of formalism. This does not prohibit the development of rather generic data definitions which are applicable for defining aspects of the information systems. However, since the information systems must be capable of describing any application area and of handling information about everything in the application areas, the language constructs for defining information systems have to be just as general as those for an arbitrary application area.

Appendix I

Additional guidelines for data designs

(This appendix does not form an integral part of this Recommendation)

I.1 General

In the following, we will give some practical guidelines for data design, without going into the theoretical and philosophical aspects of data definitions. The guidelines are divided into three classes:

A Design of classes (see I.2)

B Design of identifiers (see I.3)

C Derived data and level of detail (see I.4)

The screen layouts and implied data definitions are provided as examples only. Object class labels are underlined, attribute labels are not. If the class labels are put at the same line, the objects can exist independently of each other, but there can be a reference from an object in the leftmost class to objects in the rightmost class. If the rightmost class label is put at the line below that of the previous class, then the line shift indicates containment. Containment of attributes within object classes and within attribute groups are similarly indicated by lineshifts.

I.2 Design of classes

A1 Existential independence. The identification of the existence of objects must be done by observing which can exist independently of each other.

Example

A Subscriber line is usually associated with a Telephone number. In this example we can assume that there can be only one Telephone number for each Subscriber line and vice versa. Then it is tempting, but discouraged, to identify the Subscriber line by using the Telephone number as an identification attribute:

Subscriber line

Telephone number

06 809100

However, the Subscriber line can be allocated (and must be identified) prior to the allocation of the Telephone number. Therefore, the Subscriber line and the Telephone number should be treated as two independent, but related objects classes. The relation can have the cardinality 1:1:

Subscriber line	*Telephone number*
Name	Name
1234	06 809100

Motivation for this choice can be found when studying the life history of the object instances. If the Subscriber line was identified by the Telephone number only, as shown in the previous example, we would have to introduce unnecessary dependencies, sequences and renaming work in the routines for administering the telecommunication network.

A renaming of the Subscriber line when a Telephone number is allocated to it or is released from it, can also destroy the possibility to keep record of the history of the objects.

Further motivation for the above choice can be found when studying the references to the objects. E.g. if a reference is stated (e.g from Cable pair to the Name of Telephone number), then the renaming (of the Subscriber line, identified by different Telephone numbers), can introduce much manual or automatic work to update the cross-references. Note that the above discussion does not prohibit the existence of a direct derived relation from the Telephone number to the Cable pair(s) in the Subscriber line. See example C3.

From the above concerns we recommend not to make the reference from Cable pair to Telephone number a primitive (underived) reference:

Telephone number	*Subscriber line*	*Cable pair*
Name	Name	Name
06 809100	1234	A-B 1 15 B-C 3 1

A2 Attributes versus object classes. When we assign attributes to objects, we have to ensure that this information cannot exist independently of the object.

Example

If for each Circuit we want to register which Circuit group it is related to, and several circuits can be related to the same Circuit Group, then the Circuit group has to be considered a separate object class a), and not to be an attribute only b).

a)

Circuit	*Circuit group*
Name	Name
A-B 1	A-B 1
A-B 2	A-B 1
A-B 3	A-B 2
B-E 1	B-E 1

b)

Circuit	
Name	Circuit group
A-B 1	A-B 1
A-B 2	A-B 1
A-B 3	A-B 2
B-E 1	B-E 1

Note that in alternative a) we can look up a Circuit group directly without (having to state) a search through the circuits.

A3 Decoupling. Different facts should be kept apart in separate attributes and not be coded in the same values.

Example

Suppose that persons can change over time from being short to tall, and change between being thin and thick. Then this information should be coded into two different attributes:

Height (with value set):

– short;

– tall.

Figure (with value set):

– thin;

– thick.

This will allow one piece of information to be changed and separate logic to be written without affecting the other. A more unfortunate data design would be to have a single attribute:

Size (with value set):

– small (short and thin);

– round (short and thick);

– long (tall and thin);

– large (tall and thick).

A4 Interrelated application areas. It is of great importance to identify and delimit the right application area for the Application Schema; however, it is not prohibited to define and to interrelate data from different application areas.

Example

Suppose we want to define data about traffic routing in the telecommunication network. The object classes from the network application area can be Destination, Route, Circuit group, etc. This does not prohibit defining an object class "Parameter set" from the domain of the information system or organization administering the network. The "Parameter set" can be identified by a time stamp and/or a user name. The objects from this class can identify the combinations of Destination, Route, Circuit group, etc., which are put into operation or are registered simultaneously.

Note that the entire information system can be defined this way, including information handling processes, information queues, input-output relations, information flow, etc.

A5 Number of classes. By replacing one class by several object classes you put more constraints on the populated data; by merging several classes into one class you can relax on the constraints.

Example

a) If Telephone number and Subscriber line are merged into one class Number, and the reference between the two classes is replaced by a recursive reference from Number-to-Number, then the end user is free to choose if he will relate Cable pairs directly to (Telephone) Number or indirectly via. (Subscriber line) Number and the recursive relation.

b) If both Telephone number and Subscriber line are kept/introduced as separate classes, then the end user is constrained to relate the objects in the intended way.

We will make no recommendation as to which choice to make. Either alternative can be wanted in different cases. However, the HMI designer should be conscious about the implications of his designs.

A6 Merged presentations. The class definitions should be made such that the required objects can be conveniently presented in an integrated list.

Example

A company has two kinds of customers, Persons and Organizations. Then the only available presentation of all customers is:

Company	*Person*	
Name	Name	
	First name	Family name
A	C	A
	H	A
	B	B
	R	J
	Organization	
	Name	
	AA	
	BB	
	CC	

Example

If the company wants to create an integrated ordered list of all its Customers, then this can be done as follows:

Company		*Person*	
	Customer		
A	1	C	A
		Organization	
	2	AA	
	3	BB	
		Person	
	4	H	A
	5	B	B
	6	R	J
		Organization	
	7	CC	

Note that the attribute labels are suppressed, only the object class labels are shown.

We see that the list becomes complex, due to the different designs of identifiers of Persons and Organizations. If Persons and Organizations are merged into one single class PersOrg, then the problem disappears. This also requires that First name and Family name are merged into one long attribute only.

Example

Company		*PersOrg*	
	Customer		
A	1	C	A
	2	AA	
	3	BB	
	4	H	A
	5	B	B
	6	R	J
	7	CC	

All the above examples are valid choices. However, the designer should be aware of the user needs and implications of either choice.

I.3 Design of identifiers

B1 Local identifiers. If one object instance is contained in a superior object, then this subordinate object is identified relative to the superior object.

Example

The Circuit object class is defined (in this example only) to be subordinate to the Circuit group object class. Therefore, in this example a subordinate Circuit cannot exist independently of a superior Circuit group.

Circuit has the identifying attribute Name. Circuit group has a different identifying attribute also called Name.

The Circuit Name has numerical values within the superior Circuit group. The Circuit group Name has alphanumerical values. For example:

Circuit group	*Circuit*
Name	Name
A-B 1	1
A-B 2	2
A-C 1	1
B-E 1	1

Note that there are two circuits in Circuit group A-B 1 and no circuit in Circuit group A-C 1. Hence, the presentation is a tree and not a flat table.

B2 Independent identifiers. If one object class is not defined to be subordinate to the other, but is related to the other (perhaps at the same level), then the identifying attributes also have to be independent of each other.

Example

Circuit group	*Circuit*
Name	Name
A-B 1	A-B Z1
	A-B Z3
A-B 2	A-B Z2
A-C 1	
B-E 1	B-E Z1

Note in the above example that:

– the Circuit Name has to be made unique by adding the names of the two connected exchanges, e.g. A-B;

– in principle the two exchanges in the Circuit Name could be different from those in the Circuit group Name;

– the numbers in the Circuit Name are different from those in the Circuit group Name, even if they coincidently may look similar;

– the numbers in the Circuit Name are no more assigned locally to the Circuit group Name; hence, the Circuit in the third line in the example had to be assigned a number different from that of the previous example in order to distinguish the circuits in the first and third line;

– a Z is added to tell that all these circuits are traffic circuits; even and odd numbers tell the traffic direction.

Note that defining data locally to other data (e.g. B1) can be very beneficial if the objects can exist only when the superior object exist. However, in B2 the name of a circuit has to be changed when its usage is changed, which makes it difficult to trace the history of a certain object. See also B8 and B9.

B3 Scope of identifiers. Even "global identifiers" are local to some "system" object. The "system" object should be identified and the scope of the "global identifiers" should be settled, in order to be able to communicate data with the environment of the system.

Example

This example illustrates two perspectives. First an Exchange is chosen to be 'the system'. Second, the entire Network is chosen to be the 'system'. The exchange becomes a component in this larger 'system'.

In the first option all objects were observed from one/each Exchange only, then these objects (e.g. Circuit groups) can be identified locally to this Exchange. Hence, there is no need for coordination to the assignment of identifiers of objects outside the Exchange. In the second option Circuit group is labelled and observed directly from the 'Network', then this class must be different from the Circuit group class seen from Exchange, since no class or instance can be labelled or identified and seen from two or more superior object classes or objects. The reference from the Network Circuit group to the Exchange Circuit group is called a Terminated circuit group.

System

Name	*Exchange*		
	Name	*Circuit group*	
		Name	*Circuit*
			Name
Network	A	B 1	1
			2
		B 2	1
		C 1	
		E 1	1

	Circuit group		
	Name	*Terminated circuit group*	
		Name	
		Exchange	Circuit group
	A-B 1	A	B 1
	A-B 2	A	B 2
	A-C 1	A	C 1
	A-E 1	A	E 1

Alternatively, the references to Exchange Circuit groups could be made by using 'backward local naming':

Network

Name	*Circuit group*		
	Name	*Terminated circuit group*	*Exchange*
		Name	Name
Network	A-B 1	B1	A etc.

This presentation can be derived from the same Application Schema. However, while the Exchange name in the previous example is selected from the Terminated circuit group, this Exchange Name is selected from the Exchange itself. This allows for the addressing and modification of the Name of the Exchange (not only the reference to it, as in the first example) and modifications of other information about the Exchange in this presentation.

B4 Use of subclasses. It is recommended not to use the notion of subclass, but to consider different classes to have different instances and different identifications.

Example

Company		*Person*	*Individual*
Name	*Customer*		
	Name	Name	Name
IBM	1	John Smith	A
	2	John Smith	A
	3	Bill Jones	B
ND	1	Bill Jones	B

The identifiers of the class Person do not apply for all objects of the class Individual, which may also include Organizations, etc. Therefore, the Individual is assigned a Name which is harmonizing the identifier values for all its object instances. However, for each Person it is required that there is an Individual. But, it is not stated that the Person and the Organization Roles of an Individual exclude each other.

Customer can be considered a view of Person from the point of view of one Company, as long as there is registered one Customer for each Person only. In this perspective, it could be tempting to use the Person Name to identify the corresponding Customer. However, if we allow the Customer and the Person to exist independently of each other, then they should be assigned different identifiers, even if they may be connected by a 1:1 reference. Ref. also example A1.

If we extend the definitions to allow several customers for each Person, then it would be very unfortunate if we previously had used the same identifier for both classes and would have to rename all customers. See Customer 1 and 2, corresponding to the one Person John Smith.

If we allow each Customer to be seen from a separate (daughter) Company (after a merger), then the need for identifying the customers independently becomes obvious. See Customer IBM 3 and ND 1 for the same Person Bill Jones.

We recognise that it is important to design the data in such a way that they are prepared for evolution, and that they do not cause confusion.

Note that according to the above line of thinking, Person cannot be defined to be a subclass of Individual, because the Name of Individual is not inherited. However, if the inheritable attributes of Individual are collected in a contained class (cardinality (1,1) of Individual, then Person can inherit from this contained class. Similarly for Customer and Person.

NOTE – The notion of subclasses/inheritance is defined and used in different ways in different approaches:

a) A statement to copy specifications from one class to another, while the instantiations of the subclasses and superclasses are independent; this is the chosen approach in the Data Oriented HMI Specification Technique.

b) Each instance of the subclass is also a member of all its superior classes, in the set theoretic way.

c) One individual (e.g. John Smith) of a (super)class (e.g. Person) can belong to several of its subclasses (e.g. Customer and Employee) simultaneously.

d) Membership in one subclass (e.g. Woman) can exclude membership in another subclass (e.g. Man) of a common superclass (e.g. Person).

To express the alternatives, a formalism would need different statements or combinations of statements.

B4 recommends not to use subclasses for object classes when specifying HMIs. However, subclasses for copying specifications can be beneficial for attributes.

In a) we can easily copy too much; if not artificial classes to be copied are created.

In b) we are not able to relate two instances of a subclass to one instance of the superclass.

c) can better be treated by the reference notion which is missing in many object oriented approaches.

d) requires conditional statements, which will have to be elaborated in future extensions of the formalism.

B5 Attributes locally to object classes. If one attribute is used within one object class only, then there is no need for the coordination of the assignment of attribute values to objects outside this class. At the HMIs we will not allow identifiers to span over several object classes.

Example

If Circuit group and Circuit are two different classes, then their identifying attributes should be independent of each other, allowing the same values to be used for identifying different objects in the distinct classes.

Circuit group	*Circuit*
Name	Name
A-B 1	A-B 1
	A-B 2
A-B 2	A-B 3
A-C 1	
B-E 1	B-E 1

Note that when making references between two independent objects, which are not related by containment, then the full Name of the referenced object has to be provided, even if a part of the referenced Name is identical to the Name of current object.

B6 Complex identifiers. In B1 we have seen objects being named locally to a superior object. However, we should recognise that the identification mechanisms can be more complex.

Example

A Circuit group can be identified by the two exchanges (in alphabetic order) which it connects.

Circuit group	*Exchange*
Name	Name
A-B 1	A
	B

We see that information can (and sometimes have to) be seemingly redundantly defined.

Note that we have suppressed the reference attribute from Circuit group to Exchange, because a separate column for the reference attribute would not contain any new information added to the column for the Name of the referenced Exchanges. This suppression may have consequences for the end user dialogue. See end of B4.

B7 Attribute groups. The identifying attribute of an object can be an attribute group. At the HMI we are not restricted to identifying objects by using one attribute only.

Example

System

Name	*Circuit*		
	Nom		
	Exchange 1	Exchange 2	No.
PTN	A	B	1

It is recommended to sort out the different pieces of information in separate attributes, and not to code much information in one complex value.

B8 Stability of identifiers. Identifier attributes can contain data that convey information about properties of the objects, but should only contain data that cannot be changed throughout the lifetime of the object.

Example

Circuit

Name
A-B Z1

Note that contrary to B7 where the attribute Name is decomposed into parts, in this example the value of the attribute Name is decomposed into parts.

If the use of a Circuit can be changed from that of being a traffic circuit (Z) to becoming a leased circuit, a traffic circuit in the other direction, used for other services, etc., then it is very unwise to have this modifiable information in the identifying attribute.

The example shows an unfortunate coding of information into the value of one single attribute, where the Z tells about the usage of the Circuit and odd and even numbers tells about traffic directions. See also B2.

B9 Instability of identifiers. Sometimes we have to include bad data designs, because of the commonly accepted usage which we are unable to change.

Example

Person

Name			
Christian name	Family name	Address	
		Street	N°
John	Smith	Kings road	11

In this case the Address of the Person can change while the Address is a part of the identifying attribute.

B10 Duplicates. Sometimes it can be necessary and convenient at the HMI to allow the registering of data which are currently not uniquely identified.

Example

Address	*Person*
Name	Name
K. str. 5	John Smith
	John Smith

At a later moment of time we can receive information that permits the entries to become unique. In order to handle text and graphics files we may have similar needs to handle significant duplicates.

Another example is the presentation of similar icons representing different exchange instances, without providing data that distinguish the exchanges on that map, except for the positioning of the icons.

I.4 Derived data and level of detail

C1 Granularity versus efficiency. When we assign attributes to objects, we have to ensure that the attributes provide the necessary flexibility and the required efficiency.

Example

a)

Circuit group

Name	*Circuit*	
	Nom	Date
A-B 1	1	900223
	2	900224
A-B 2	1	900224
A-C 1		
B-E 1	1	900224

b)

Circuit group

Name	Date	*Circuit*
		Name
A-B 1	900223	1
		2
A-B 2	900224	1
A-C 1	900224	
B-E 1	900224	1

In example a) a different Date can be registered for when each Circuit was put into operation. This provides much flexibility for the registering of individual dates, but requires much work for the registering of all the dates.

In example b) the start dates can be registered for each Circuit group only. This provides less flexibility for registering individual information for each Circuit, but requires much less work for the HMI operators.

C2 Derived attributes. The HMI data definitions shall not contain basic data only, but also all derived data shown at the HMIs.

Example

When looking up a Circuit group, it can be beneficial to the users to see the "# of idle circ." contained in the Circuit group, rather than having to search through all its circuits for those which are Idle.

Circuit group

Name	# idle circuit	*Circuit*	
		Name	Status
A-B 1	1	1	
		2	Idle
A-B 2	0	1	
A-C 1	0		
B-E 1	1	1	Idle

C3 Derived references. The HMI data definitions can contain derived references.

Example

Telephone number	*Subscriber line*	*Cable pair*
Name	Name	Name
06 808011	9876543	A-B 31 B-C 15
	Cable pair	
	Name	
	A-B 31 B-C 15	

See more details in A1.

A derived reference is a reference which is derived from other data; e.g. if there is a path class-1, ref-2, class-2, ref-3, ..., class-n, then there can be introduced a direct derived reference ref-1-n from class-1 to class-n (if so defined), and similarly for the corresponding instances.

Note that derived data do not have to be physically stored in the internal layer, but all data used in the external layer have to be defined in the application layer. Therefore, the three layers behave as a consistent whole.

C4 Derived objects. The HMI data definitions can contain derived objects.

Example

Link	*Circuit*
Name	Name
A-B	A-B 1 A-B 2 A-B 3
A-C	
B-E	B-E 1

The Link object class can be introduced as a derived object to provide an overview of all circuits between two exchanges only. We assume that there can be only one Link between two exchanges.