International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.601

(02/2007)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Middleware – Distributed processing environment

Data architecture of one software system

ITU-T Recommendation Z.601

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
Extended Object Definition Language (eODL)	Z.130–Z.139
Testing and Test Control Notation (TTCN)	Z.140–Z.149
User Requirements Notation (URN)	Z.150–Z.159
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Distributed processing environment	**Z.600–Z.609**

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Z.601

Data architecture of one software system

Summary

ITU-T Recommendation Z.601 identifies a set of data structures and formats of one software system.

These data forms appear at the various interfaces to and media of the system and comprise intermediate forms for transformations between the external forms. The data forms are needed within one system, and they are not abstracted away from the system over several systems.

This Recommendation identifies data schemata that may be used to define interfaces between software components. However, this Recommendation defines no software architecture.

Source

ITU-T Recommendation Z.601 was approved on 13 February 2007 by ITU-T Study Group 17 (2005-2008) under the ITU-T Recommendation A.8 procedure.

Keywords

Architecture, data, form, format, framework, layer, population, process, schema, structure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. 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 Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

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

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Introduction

The primary users of this Recommendation will be software developers who design data definitions of a system and its interfaces.

Data architects and systems planners may use this Recommendation to coordinate definitions of interfaces between systems.

Some of the data definitions may provide end users' understanding of the system and its functionality. Hence, these definitions will provide a kernel of a contract between purchaser and developer organizations.

Formal language designers may use this Recommendation to identify the scope of their notation and identify features needed to cover a certain application domain.

Annex A provides requirements on schema languages.

Appendix I provides an introduction and context of the Recommendation, where a system is defined in the context of systems planning of several systems. The reader is advised to read this appendix before he reads clause 6.

Appendix II provides a comparison with [b-ITU-T M.3020].

ITU-T Recommendation Z.601

Data architecture of one software system

1 Scope

This Recommendation identifies a set of data structures and formats of one software system.

These data forms appear at the various interfaces to and media of the system and comprise intermediate forms for transformations between the external forms.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Z.351] ITU-T Recommendation Z.351 (1993), *Data oriented human-machine interface specification technique – Introduction*.

3 Definitions

This Recommendation contains a set of definitions. The following terms appear in this Recommendation. They are either defined explicitly, or by simple inference from the definitions. The terms are meant to be used within the context of the data architecture, and may not have the same meaning outside this context.

application area: See [ITU-T Z.351].

application layer: See [ITU-T Z.351].

application population: See [ITU-T Z.351].

application process: See [ITU-T Z.351].

application schema: See [ITU-T Z.351].

concept layer.

concept population.

concept process.

concept schema.

contents layer.

contents population.

contents process.

contents schema: See [ITU-T Z.351].

distribution layer.

distribution population.

distribution process.

distribution schema.

external layer: See [ITU-T Z.351].
external population: See [ITU-T Z.351].
external process: See [ITU-T Z.351].
external schema: See [ITU-T Z.351].
external terminology layer.
external terminology population.
external terminology process.
external terminology schema.
internal layer: See [ITU-T Z.351].
internal population.
internal process.
internal schema.
internal terminology layer.
internal terminology population.
internal terminology process.
internal terminology schema.
layer: See [ITU-T Z.351].
layout layer.
layout population.
layout process.
layout schema: See [ITU-T Z.351].
physical layer.
physical population.
physical process.
physical schema.
population: See [ITU-T Z.351].
process: See [ITU-T Z.351].
schema: See [ITU-T Z.351].
system.
system management layer.
system management population.
system management process.
system management schema.
terminology layer.
terminology population.
terminology process.
terminology schema.

4 Abbreviations

This Recommendation uses the following abbreviations:

AL Application Layer

AP Application Population

Ar Application Process

AS Application Schema

CL Contents Layer

CP Contents Population

Cr Contents Process

CS Contents Schema

DL Distribution Layer

DP Distribution Population

Dr Distribution Process

DS Distribution Schema

EL External Layer

EP External Population

Er External Process

ES External Schema

eTL external Terminology Layer

eTP external Terminology Population

eTr external Terminology Process

eTS external Terminology Schema

IL Internal Layer

IP Internal Population

Ir Internal Process

IS Internal Schema

iTL internal Terminology Layer

iTP internal Terminology Population

iTr internal Terminology Process

iTS internal Terminology Schema

LL Layout Layer

LP Layout Population

Lr Layout Process

LS Layout Schema

OL Concept Layer

OP Concept Population

Or Concept Process

OS Concept Schema

PL Physical Layer

PP Physical Population

Pr Physical Process

PS Physical Schema

SML System Management Layer

SMP System Management Population

SMr System Management Process

SMS System Management Schema

TL Terminology Layer

TP Terminology Population

Tr Terminology Process

TS Terminology Schema

5 Conventions

S Schema

P Population

6 Data architecture

6.1 Three-schema architecture

The three-schema architecture comprises the following kinds of schemata:

- external schemata;
- application schema;
- internal schemata.

For one system instance, the three-schema architecture provides one centralized application schema, which may support several external and internal schemata. For n external and m internal schemata, the three-schema architecture provides (n+m) mappings between schemata.

The application schema of a system instance defines the data classes, constraints and derivations which have to be enforced for any data within the system instance, independently of which external or internal medium they may appear on.

Figure 1 depicts example schemata and corresponding populations in the three-schema architecture.

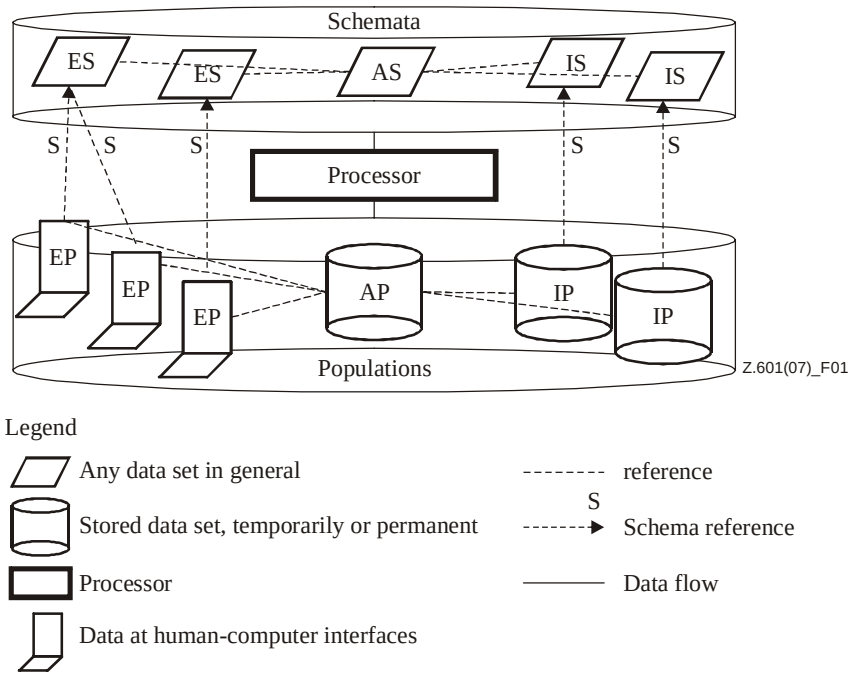


Figure 1 – Example use of the three-schema architecture

Note that the three-schema architecture does not state which external population and corresponding schemata are used to input data to which internal population and corresponding schema. Hence, internal forms are decoupled from external forms of data. Also, each external and internal schema is decoupled from any other external or internal schema, respectively. The application schema provides the only link between external and internal forms.

The correspondences between schemata, processors and populations may be conveyed through the notion of layers:

- external layer;
- application layer;
- internal layer.

This is illustrated in Figure 2.

Note that the definition of the application schema presupposes that every data instance is transformed via the application layer. Strictly speaking, the external form of data is one instance, the application form is a second and the internal form is a third. And these may be copied into several external and internal populations, creating still more instances, all originating from one input instance, e.g., in the external layer.

Note that the three-schema architecture is a compiler kind of architecture, where presentation forms of data at the external and internal layers are compiled via an intermediate application layer. This compiler feature of the architecture may be used to nest the architecture.

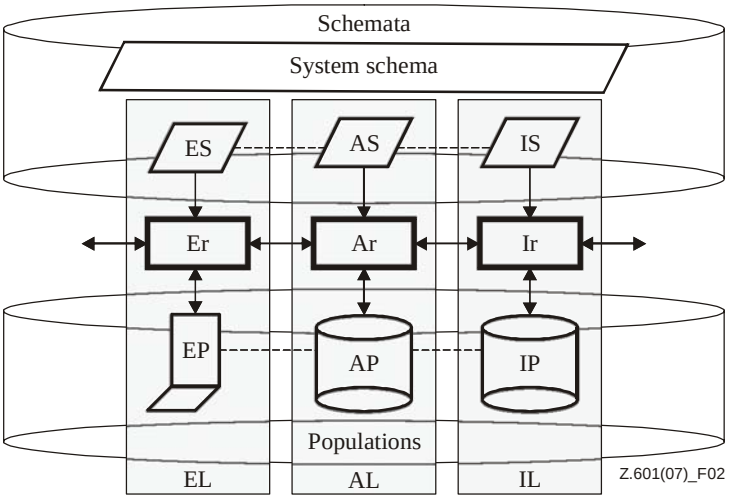


Figure 2 – Layering

In order to cover the coordination between the layers, we introduce some extra notions:

- System management schema, containing – except the external, application and internal schemata:
  - security data, including access control data;
  - system directory data, including data for configuration control.
- System management processor, including the external, application and internal processors, controls their interoperation and provides security and directory services.
- System management population: provides data instances for security and directory services.

Note that the system management processor and population are not depicted in Figure 2. With the system management schema, we have in fact a three plus one-schema architecture.

We also observe that the processor of the system instance is split into specialized processors per layer.

6.2 Seven-schema architecture

The seven-schema architecture provides a detail of the three-schema architecture. Each layer of the three-schema architecture is split into two or three sub-layers. This is depicted in Figure 3.

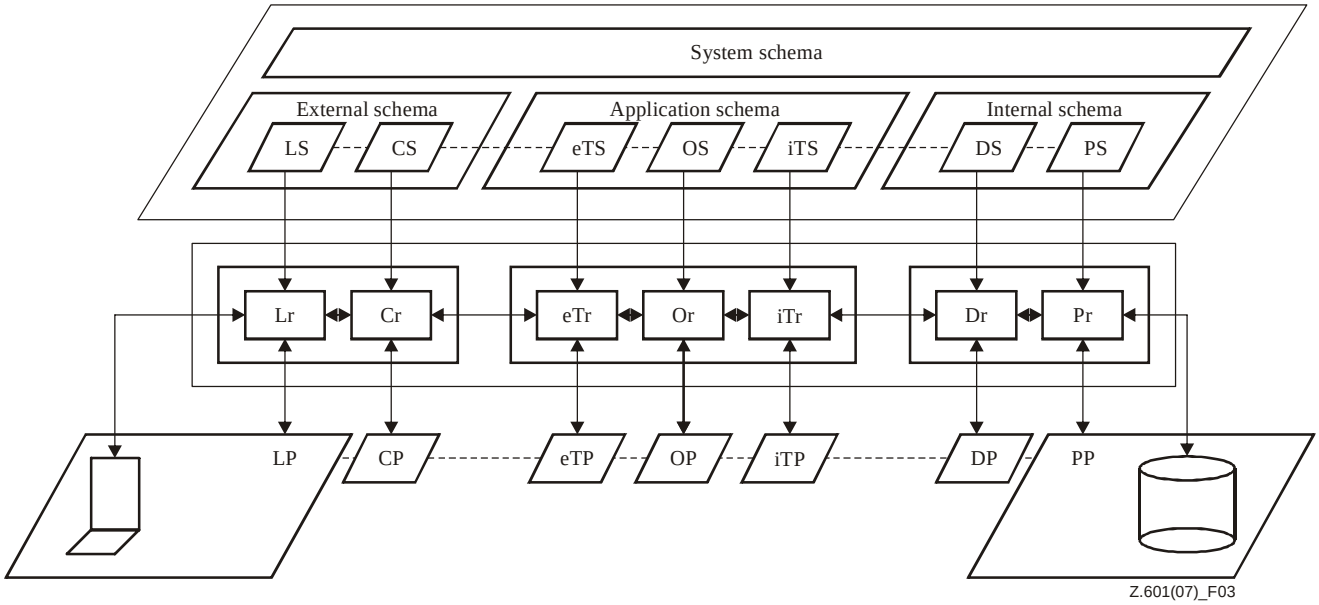


Figure 3 – Data transformation architecture

The seven-layer architecture describes how data can be transformed from one format and terminology into another format and terminology.

The schemata of the seven layers are as follows:

- The external schema is composed of:
  - layout schema, which defines the way data are presented to human users;
  - contents schema, which defines the contents and structure of the selected data and permissible operations on these data.
- The application schema is composed of:
  - external terminology schema, which defines the common terminology and grammar for a set of external schemata;
  - concept schema, which defines the common structure, constraints and derivations of data, common for all terminologies supported by the system – defined in each external and internal terminology schema;
  - internal terminology schema, which defines the common terminology and grammar for a set of internal schemata.
- The internal schema is composed of:
  - distribution schema, which defines the sub-setting of one internal terminology schema for one medium and permissible operations on these data on this medium;
  - physical schema, which defines the internal storage, accessing, implementation and communication of data and their behaviour.

There will be corresponding processors and populations. In addition, come the system schema, processor and population.

The layout schema defines fonts, colours, placements, buttons, lines, shapes, etc., as seen by the human users.

The contents schema defines the grammar of the presented statements.

The external terminology schema defines the alphanumerical or graphical language being used.

The concept schema defines the ideas or notions being managed by the system instance.

The internal terminology schema defines the language being used for storage or communication of data.

The distribution schema defines the contents of messages being communicated to other systems or storage media.

The physical schema defines the actual layout or encoding on the non-human media.

The seven-layer architecture is symmetrical around the concept schema:

- The layout schema corresponds to the physical schema.
- The contents schema corresponds to the distribution schema.
- The external terminology schema corresponds to the internal terminology schema.

The external and internal terminology schemata are collectively called terminology schemata.

The external and internal schemata are collectively called presentation schemata.

The mappings between these schemata are illustrated in Figure 4.

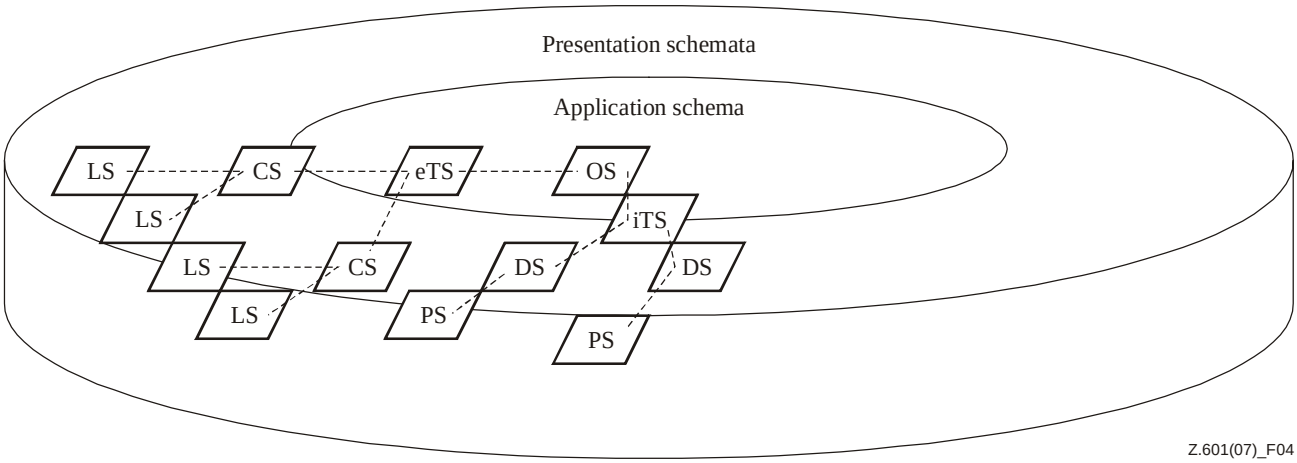


Figure 4 – Example data flow between layers

Figure 4 depicts the permissible data flow between layers of the data architecture.

Note that the two-way flows are stated without any mentioning of processes or processors.

However, these are not the only possible flows:

- If one external terminology schema is stated for all concepts, and all constraints and derivations are also stated in this external terminology schema, then this external terminology schema may replace the concept schema altogether.
- If the internal media are using the same terminology as (one of) the external media, the internal terminology schema may be replaced by this external terminology schema as well.

The effects of the two conditions are illustrated in Figure 5.

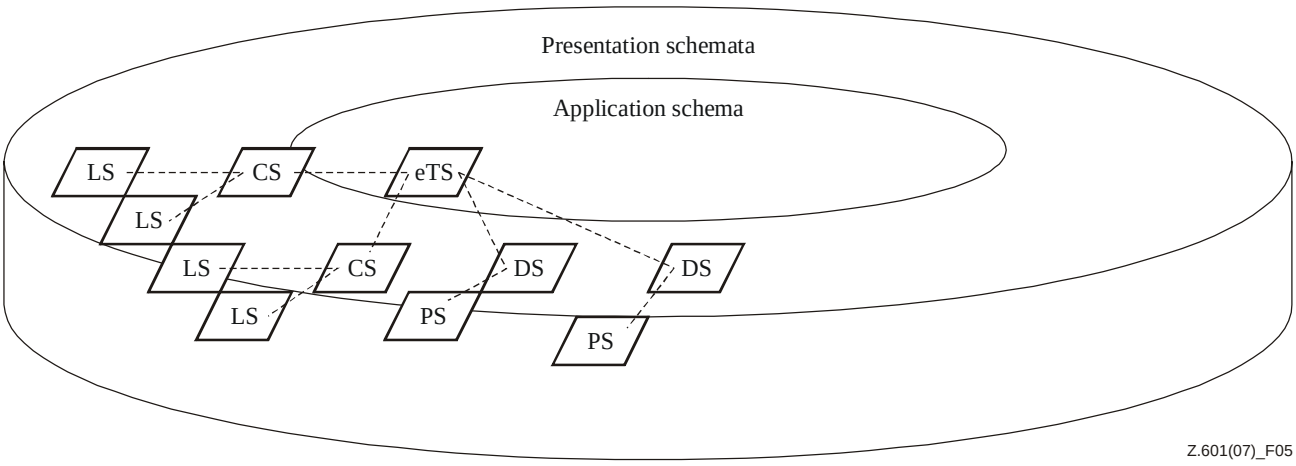


Figure 5 – Prescription of simplified data flow

The system instance may be further degenerated, but with the loss of external terminology schemata there will not be any data instances to enforce consistency over. Hence, this will be no more one integrated system instance, but may be a function of some other system instance.

Note that other terminologies than the above are frequently used: the layout schema may be called a GUI or panel. The contents schema may be called a view. The application schema may be called a data structure, etc. However, these alternative notions may not satisfy the requirements expressed in Annex A.

6.3 Communication and distribution

For two communicating system instances to be able to interpret the communicated data correctly, they have to have identical schemata for the communicated data. The same applies for exchange of data between two communicating software components. Hence, the layers of the seven-schema architecture do not identify what could be one software component, as they all have different schemata.

Two communicating software components will have identical schemata for the communicated data, and transformations between schemata will take place inside the components. This is illustrated in Figure 6.

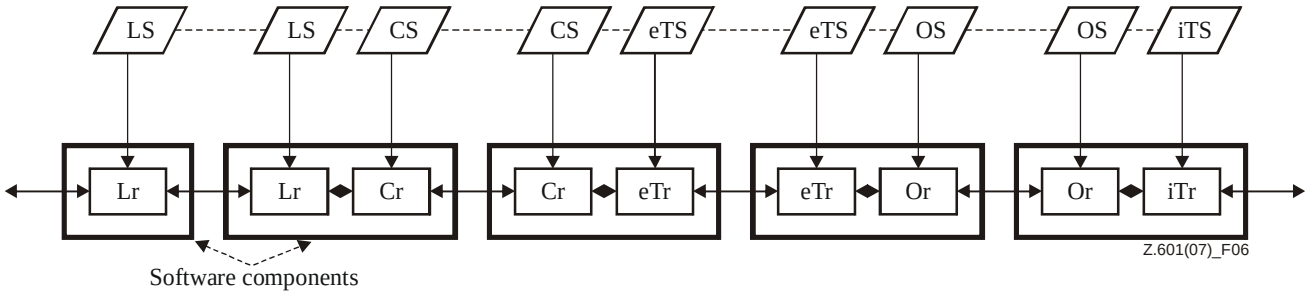


Figure 6 – Candidate software components

Implementations to and in the internal layer is not illustrated in Figure 6. However, the same principles apply.

Figure 6 shows that a processor must be able to communicate with a processor of the same kind, e.g., eTr must be able to communicate with eTr, and Or must be able to communicate with Or. Also, transformations between schemata may not take place in every component. See, for example, the Lr processor in Figure 6.

In Figure 7, eTr of one component communicates directly with eTr of another component. What data is communicated is stated in the distribution schemata. It will depend on the transaction handling of the two eTr components if they implement one system instance or not.

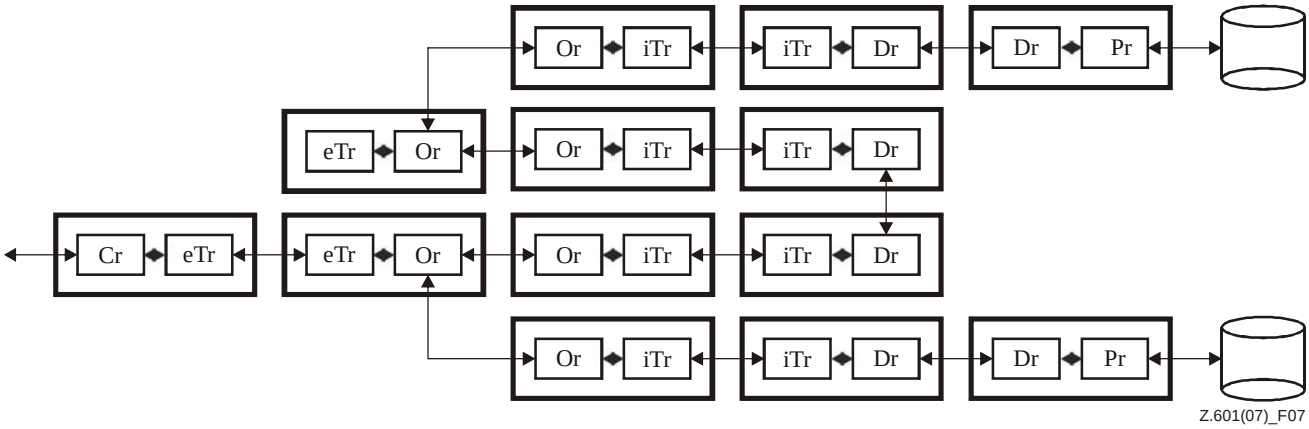


Figure 7 – Distribution of processing

If the components implement one system instance, their data instances may be partitioned horizontally or vertically. If the partitioning is vertical, their schemata may only be identical for the communicated data.

The components may run on the same hardware. If so, communication between identical kind of processors may happen directly. If not, the communication may be stated via appropriate distribution and physical schemata.

Note that the distribution schema is the only means to state what data shall be communicated where.

If data of one or more sub-layers of the application layer of one system instance is partitioned over several processors, then this system instance is called an integrated distributed system instance.

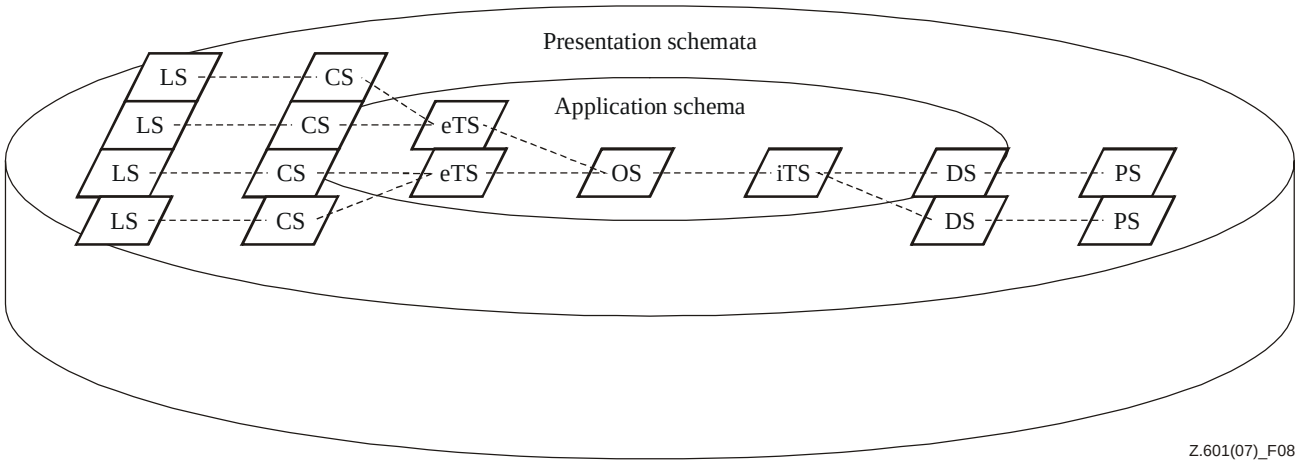


Figure 8 – Decoupling of layers through a centralized application schema

The purpose of the insulation functionality is to provide modeless-ness. Any screen picture may be used to update a particular piece of information (in the application layer), as long as this information appears in the screen picture and allows for the appropriate operation on the data instance. Some screens may be tailored for this function, but the user is not forced to use particular screens or dialogue sequences, e.g., through using a particular order entry process. Hence, the data architecture provides decoupling of layers.

Even if the data architecture functions as an insulator on the original source, form and destination of information, this may be worked around by defining particular data on this information in the application schema or population. An example of this is handling of an order of a particular product. The product type requires certain data to be provided, and the order is not accepted until the appropriate data are provided. This can be expressed as constraints on the associations between the product type and order (in the application layer), and need not be designed as an application dependent process for order entry. For a certain product type – within an order – the use of certain internal interfaces is prescribed; this can also be expressed as a method on the data (in the application layer), and need not have a particular work flow process. This way, the application schema and population may accomplish the coupling that the architecture does not provide. The actual provision of the last example will be expressed through the distribution schema for the actual data.

6.4 Nesting

The data architecture provides a translation and reorganization of any data. Hence, the architecture may be used to manage the schema data as well as instance data. This is illustrated in Figure 9.

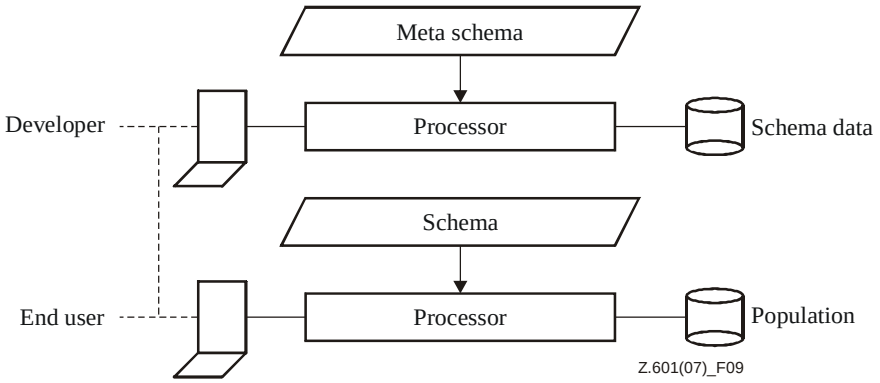


Figure 9 – Recursive use of the data architecture

Figure 9 shows that the data transformation architecture may be used to implement a repository of specifications (external layers) and their implementations (internal layers) of an application, e.g., of an operation support system for telecommunications management.

Also, the data architecture provides a means to compile or interpret the stored schema data in the repository into executable code of the application. This is illustrated in Figure 10.

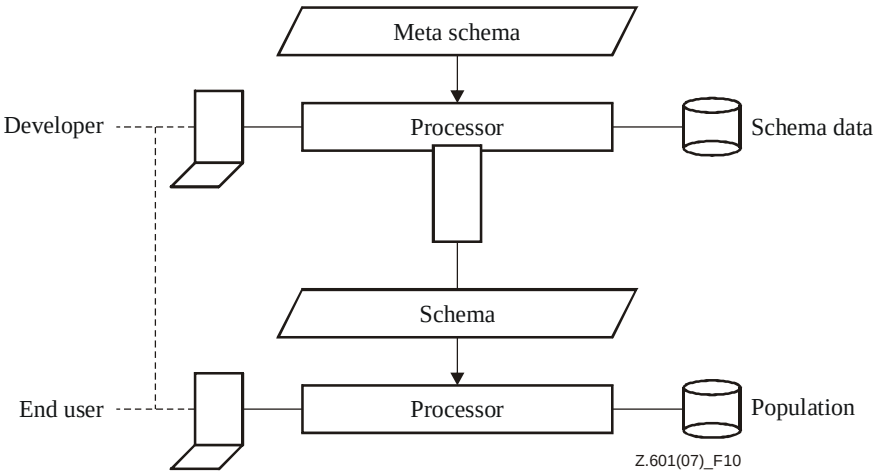


Figure 10 – Active repository

Or, the specifications as seen by the system developer may be transformed directly into executable code as may also be accomplished by Figure 10.

Note that Figure 9 allows for ordinary end users accessing the repository, and Figure 10 allows for using the repository for active help to access the application on the population data.

The implication of Figures 9 and 10 is that the format of a schema is not well-defined by referring to the layer only. In addition, we have to refer to the form of the layer information, e.g.:

- layout form of layout schema;
- contents form of layout schema;
- ..
- physical form of layout schema;
- etc.

We get a matrix of forms for the various schemata, as illustrated in Figure 11.

LLS	LCS	LeTS	LOS	LiTS	LDS	LPS
CLS	CCS	CeTS	COS	CiTS	CDS	CPS
eTLS	eTCS	eTeT	eTOS	eTiTS	eTDS	eTPS
OLS	OCS	OeTS	OOS	OiTS	ODS	OPS
iTLS	iTCS	iTeTS	iTOS	iTiTS	iTDS	iTPS
DLS	DCS	DeTS	DOS	DiTS	DDS	DPS
PLS	PCS	PeTS	POS	PiTS	PDS	PPS

Figure 11 – Nesting of the data architecture

The upper row of Figure 11 shows the formats of the schema as presented to human users.

The lower row shows the physical formats of the schemata, e.g., their executable form.

From the above illustrations in Figures 9 through 11, we realize that the data architecture may be used recursively. And, there may be no absolute distinction between schema data and instance data, as they are only so relative to each other. This means that:

- data types (such as numbers) in the schema of an application may serve as classes for other data (such as length) in the same schema;
- data instances (such as product types) in one population of an application may serve as classes for other data (such as install base) in the same population.

Annex A

Requirements on schema notations

(This annex forms an integral part of this Recommendation)

A.1 Introduction

The external terminology schemata define the terminology and grammar of one application system. Therefore, one such schema should be clearly defined before development of any other part of the application system, i.e., before defining the other schemata.

Hence, the requirements on the notation for the external terminology schemata come first in this annex and should be understood before requirements on other schema notations are discussed.

Note that the requirements on the physical schemata, i.e., the external layout schema and the internal physical schema are not explored in detail in this annex.

The requirements are presented in the following sequence:

- external terminology schema;
- concept schema;
- contents schema;
- layout schema;
- internal terminology schema;
- distribution schema;
- physical schema;
- system management schema.

A.2 Requirements on notations for the external terminology schemata

A.2.1 Introduction

The external terminology schema should be capable of expressing the syntactical richness of elementary statements and their constituents as encountered at the human computer interface of one system. The following requirements may all be considered means to this end. However, the full capability to the human user is only provided by the addition of the features of the contents and layout schemata, as indicated in the text. Also, features of the system management schemata are needed for management purposes, and the internal schemata are needed for implementation.

A.2.2 Requirements

1) The contents of an external terminology schema are recursively made up of lists of elementary statements and their constituents, which may contain references to other statements or constituents.
    a) The external terminology schema is itself a node in a data tree, which also contains its elementary statements and other constituents.
    b) The purpose of the external terminology schema is to prescribe permissible contents of its external terminology populations.
    c) The purpose of restricting the external terminology schema to elementary statements and their constituents is that the creation of compound statements is deferred to the contents schemata.
    d) The expression 'elementary statements' is meant to refer to data structuring notions, like objects and attributes, and is not meant to require use of sentential logic; the expression 'and their constituents' is meant to indicate value syntax and behaviour expressions.

e) Lists within the external terminology schema are used to express alternatives, i.e., disjunctions, of what is permissible in the external terminology population.

f) Lists within lists are used to express context.

2) The contents of any external terminology population is homomorphic to its external terminology schema.

a) For each node in the external terminology population, there is an identical node in the external terminology schema.

b) For each node in an external terminology schema, there may be zero or more nodes in its external terminology population. This instantiation is made by copying class labels and references between classes into the external terminology population.

c) Any node in the external terminology schema is called a class relative to its corresponding nodes in the external terminology population, which are called instances relative to their class. The nodes are classes and instances relative to each other, and are not marked as such by any reserved words.

d) If there is a S(chema) reference from a node to another node, then this other node is called a schema relative to the first node. The schema node contains the classes of the instances contained in the first node.

e) If there is a P(opulation) reference from a node to another node, then this other node is called a population relative to the first node. The population node contains instances of the classes contained in the first node.

f) Given an instance and its corresponding class; then the homomorphy requirement implies that the superior node of the class is itself a class of the superior node of the instance.

g) If a reference is made between two instances, then the homomorphy requirement implies that a corresponding reference is made between the corresponding classes.

h) The purpose of this requirement is to allow end users to foresee permissible instance structures (just by making copies of the schemata) when they see the class structures, and vice versa.

i) One implication of this requirement is that the user can see no difference between a class and an instance label, when they are observed in isolation. Only their recursively superior structures and references to schemata or populations can show the different roles.

j) In predicate calculus, there is no distinction made between instances and classes; the statements are just checked for consistency or inconsistency. We make, however, a distinction between instances and classes, because we need to know what data are prescriptions for which data. But we do not make the distinction in an absolute way. Data are just instances and classes relative to each other.

3) The name label of any node in the external terminology schema can be alphanumerical or graphical.

a) The purpose of this requirement is to allow use of any character set, and the designer should be allowed to design his own characters.

b) Graphical name labels may be pixel or vector based. This is used to prescribe graphics of the instances, as the class labels are copied into each one of its instances.

4) The notation of the external terminology schema shall be postfix and have no superficial block structure.

   a) The purpose of disallowing superficial block structures is to satisfy the homomorphy requirement; i.e., the structure of the instances shall be homomorphic to the structure of its classes. Hence, extra nodes or levels among the classes are disallowed.

   b) The purpose of requiring the postfix notation is due to the homomorphism requirement, as well. If (logical) operators will not appear above or between the nodes in the instances, they cannot appear above or between the corresponding classes either. Hence, all constraints and derivations are expressed subordinate to some class node.

   c) The previous bullet explains why the external terminology schemata contains elementary statements only, and do not contain expressions that are consistent with or derivable from the elementary statements.

   d) Derived statements that are themselves elementary are, however, contained in the external terminology schemata. The derived statements may be derived by a compound function that must itself be contained in the external terminology schema.

5) Class labels are only unique within the scope of the superior class label, if a larger scope is not explicitly stated at a superior node. See c.

   a) This means that each node acts as a block for its subordinate nodes.

   b) Identical class labels can therefore be reused for different purposes within different classes.

   c) Some classes may contain further constraints (on naming), e.g., that all recursively subordinate name labels to a constrained class must be unique.

   d) The implication of this requirement of using local class labels is that references to a class must contain the path from the current node to the referenced node. Note that use of a globally unique path of class labels may not work, as the exact path from node to node is needed to express the exact scope of the reference.

   e) Note that the path of the reference between class labels prescribes the permissible paths of references between instances. For example, a path from a 'termination point' instance via its superior 'exchange' instance to another 'termination point' instance states a connection within that 'exchange'. If alternatively, the path goes via the superior 'network' of the 'exchange', the connection can be made to a 'termination point' within any 'exchange' in that 'network'. Hence, the navigation paths provide a scoping mechanism of the references.

   f) Another implication of using local class labels is that no class label has a unique identification without providing the entire globally distinguished class label, which is made up of the path of its recursively superior class labels. Hence, to state a class label without providing its context makes no sense.

   g) The superior class of a local class label is used to express the context of the local class label. Use of local class labels allows users to use their own terminology for classes and not to invent artificial labels to satisfy the naming convention of a particular specification language.

   h) As class labels are copied into becoming instance labels, use of local class labels has implications for the naming tree of the corresponding instances. Use of local class labels prescribes name bindings between the corresponding instances.

6) Class labels need not be unique within the scope of their superior class label, as use of duplicates is permissible.

   a) Duplicate class labels are distinguishable from each other, due to their position in the list subordinate to their superior class label. Also, the subordinate structure of or references from or to duplicate class labels may be different, thus helping to distinguish the class labels.

   b) Use of duplicate class labels allows users to use their own terminology of classes and not to invent artificial labels to satisfy the naming convention of a particular specification language.

   c) The handling of lists of instances, where the instances are only distinguishable by their position in the list, is another reason for allowing use of duplicate class labels. As there is no fundamental difference between instances and classes, they are treated equally. Nodes are only classes and instances relative to each other.

   d) Within the context of a certain class, duplicate class labels may be disallowed by an explicit constraint. This certain class may be the schema itself.

   e) Also, data designers may avoid use of duplicate labels without this being explicitly stated as a constraint.

7) The external terminology schema notation should allow for recursion.

   a) Any node in the population or schema may have one or more S(chema) references to any other node in a population or schema.

   b) This means that there is no strict distinction between a population and a schema, and the notations of both are identical.

   c) This also means that a schema may have a meta-schema, etc.; or a node within a schema may have a S(chema) reference to another node within this schema, or of some other schema, or of some other populations or to a node within its own population.

   d) A S(chema) reference to a superior node is used to state recursion. The contents of the referenced superior node may act as a schema of what can be contained in the current node; hence, some contents of the superior node may be copied into the current node – maybe repeatedly.

   e) Note that any reference – not only a S(chema) reference – may refer to any node, including a superior node of the current node.

   f) Any class of a schema may be instantiated into zero or more instances. If there is no explicit cardinality constraint on the class (relative to its superior class), then any number of instances may be generated relative to their superior instance.

   g) Note that if an instance is created, also all its recursively superior instances must already exist or be created.

   h) However, if an instance is created, no subordinate instances may be created, if there is no explicit constraint requesting this creation of subordinate instances.

   i) This way, the schema prescribes the existence of all recursively superior nodes of an instance, while subordinate nodes may not exist. Therefore, the schema is considered to provide an attachment grammar as opposed to a rewriting grammar. In a rewriting grammar the superior nodes are discarded, and only leaf nodes are left in the final production. In the attachment grammar the recursively superior nodes provide the required context and must exist in the final production, while the leaf nodes may not be needed.

   j) Note that while all the recursively superior nodes within the external terminology layer are required in the contents layer, some of these nodes may be suppressed in the layout layer.

8) The external terminology schema should be capable of expressing any logical, arithmetical and quantification statements.

   a) This is called the 100% principle.

   b) The purpose of this requirement is to be able to express all constraints and derivations within the external terminology schema, and not have to rely on behaviour specifications outside this schema.

9) An external terminology schema should be capable of prescribing the abstract grammar of the elementary statements and abstract syntax of the terms of an end user terminology.

   a) Each external terminology schema prescribes different end user terminologies, e.g., English, French, Chinese and graphics.

   b) An external terminology schema does not define the common concepts of different end user terminologies, but one external terminology schema may act as the source of other end user terminologies defined in other external terminology schemata.

   c) An external terminology schema may have source references to a concept schema of that external terminology schema.

A.3 Requirements on notations for the concept schemata

A.3.1 Introduction

A copy of an external terminology schema may play the role of being a concept schema of other external terminology schemata.

This means that the notation for defining any external terminology schema satisfies the notational requirements for defining any concept schema.

The concept schema (and its population) may contain more concepts than supported in some of its external terminology schemata (and their populations).

Also, the external terminology schemata (and their populations) may contain data details that are not supported by the corresponding concept schema (and its population).

Ideally, the notational requirements for the concept schemata and the external terminology schemata are the same. However, there is a trend to not put so strict requirements on the concepts schemata. Often, the concept schemata are used to state draft notions, which are not isomorphic to their external terminology schemata. This makes it easier to state concept schemata, but makes the mapping between the layers more complex and may lead to loss of functionality of and usability from the mappings.

In conclusion, a notation for concept schemata may not satisfy all the requirements on the notation for external terminology schemata.

This is not an ideal solution, but is a pragmatic one, as human conceptions of concepts are typically more vague and sketchier than the notions of data syntax to be defined in the external terminology schemata. This has the implication that concept design is not as concrete as data design, and concept designers may provide several alternatives that they believe satisfy the data requirements. Another implication may be that concepts are only used in some analysis and design methods, and are not used in the final design and implementations.

By weakening the requirements on the concept schemata, the list of requirements may not be as exclusive as for external terminology schemata.

In addition to requirements on the definition of the concepts and their associations, mappings between nodes in the external terminology schemata and concepts are needed. This mapping is here called a denotation mapping. In this text we address the mapping between data and concepts, not between data and phenomena, and not between concepts and phenomena.

A.3.2 Requirements

1) If a node in an external terminology schema denotes a concept in its concept schema, then all its recursively superior nodes shall denote a concept as well, and a path of references between concepts shall represent the node structure.
   a) Preferably, the node structure should be mirrored by an identical concept structure, such that the concept structure becomes isomorphic to the node structure; but this would disqualify all but external terminology schema notations to specify concept schemata.
   b) The weaker formulation in this requirement states that subordination in the external layer must be represented by a path of references between concepts in the concept layer.
2) If a node in an external terminology schema does not denote a concept in its concept schema, then no recursively subordinate node shall denote a concept either.
   a) This allows subordinate nodes in the external terminology schema to provide 'syntactical sugar', e.g., be helpful for identification, but do not denote any concept.
3) An attribute in the external terminology schema may denote a role (of an object class) in its concept schema.
   a) This allows value types to be represented as object classes in the concept schema.
4) Any association in the external terminology schema that denote something must be represented by a path of (one or more) references in the concept schema.
5) In one external terminology schema there may be several synonymous terms, even if each synonym normally should be defined in a separate external terminology schema.
6) Every node in the external terminology schema must denote a concept or path of concepts in the concept schema; exceptions are made for 2.
7) Except for 2, 3, 4 and 5, the graph of the concept schema should be isomorphic to the graph of any of its external terminology schemata.

A.4 Requirements on notations for the contents schemata

A.4.1 Introduction

The contents schemata define the abstract syntax of the statements presented to the human end user.

Each contents schema refers to one external terminology schema only.

Each contents schema defines a compound statement of some of the statement contained in its external terminology schema.

A.4.2 Requirements

1) The elementary statements in the external terminology schema are combined into compound statements in the contents schemata by the use of reflexive pronouns with the proper references.
   a) The references in the contents schemata shall be able to express both parenthesis structures and alternative sequences of presentation.
   b) The previous sub-bullet means that the compound statement should be able to express the following example compound statement: "Site relationship has subordinate trail (which) has subordinate trail section (which) has subordinate physical link connection which is (physical link (which) has subordinate physical link connection)". Here the globally distinguished name of the referenced physical link connection of the physical (both in parenthesis) is given by providing the identifier of the superior physical link

first. "Which" is the reflexive pronoun; note that this is used to state references only, and not to state subordination, but is here added in parentheses to ease the reading.

c) The first sub-bullet also means that the compound statement should be able to express the following compound statement: "Site relationship has subordinate trail has subordinate trail section has subordinate physical link connection which is (physical link connection has superior physical link)". Here the globally distinguished name of the referenced physical link connection will be given last.

d) The parenthesis structures shall be capable of expressing branching, like in the following statement: "Site relationship (has subordinate trail has subordinate trail section has subordinate physical link connection which is (physical link has subordinate physical link connection)) has subordinate identifier". Here Identifier is an attribute of site relationship.

e) Also the parenthesis structures should be capable of expressing delimitation of execution, e.g., traversal of a reference one way, may not automatically lead to a traversal of the opposite reference and control of consistency, even if this is required by the external terminology schema.

f) Note that the above informal examples use just one kind of parentheses, but that different kinds may be needed in the contents schema notation.

g) Note that the compound statements in the contents schemata are traversing the elementary statements in the external terminology schema in various permissible ways.

2) The paths expressed in the contents schema shall never deviate from the paths expressed in the corresponding external terminology schema.

3) The contents schema shall not define derived data that are not already defined in the external terminology schemata.

4) The notation of the contents schema should be capable of expressing recursion that does not appear in the external terminology schema, e.g., if there is a loop of associations in the external terminology schema, the contents schema may state that this loop shall be traversed recursively until no more data instance is found.

5) The notation of the contents schema should be capable of expressing logical and arithmetical conditions on the data defined in the external terminology schemata, e.g., the external terminology schema defines that site relationship has subordinate trail has subordinate length (that is given in metres). A corresponding contents schema may restrict the selected trails to be within the range 1000-2000 metres.

6) The notation of the contents schema should be capable of expressing what data will be presented in the layout layer and what will be suppressed.

7) The notation of the contents schema should be capable of expressing permissible operations, e.g., insert, delete, modify and read, on data instances prescribed by this contents schema.

A.5 Requirements on notations for the layout schemata

A.5.1 Introduction

The layout schemata define the concrete syntax of the data as they are presented to the human end user.

Each layout schema refers to one contents schema only, but there may be several alternative layout schemata for each contents schema.

A.5.2 Requirements

1) The notations for the layout schemata should provide multimedia support.

A.6 Requirements on notations for internal terminology schemata

A.6.1 Introduction

The internal layer may adapt to the technology used for communication over a telecommunication line or in a database management system. Therefore, the internal layer may not convey or support the terminology used to communicate with a human user. However, the internal layer may have to convey information needed to translate between different terminologies of different users.

One such example can be translation between codes according to ITU-T Recs M.1400, M.3100 and regional standards for management of the telecommunication network. In these cases, different object structures are used, and there is no one-to-one mapping between fields. Hence, several fields in one standard have to be communicated to provide the contents of one field in another, and only parts of the fields are needed to generate the one field.

From this explanation follows that more information may be needed in the internal terminology schemata than conveyed in the used external terminology schema, and the mappings between the schemata (via the concept schema or not) may be complex, using detailed navigation and selection of information from unpacking the value syntax of the fields.

A.6.2 Requirements

1) The notations used for the internal terminology schemata must be capable of expressing the data structures used in implementations of peer communicating systems.
2) The notations used for the internal terminology schemata may not be capable of expressing the logic and arithmetic of the application system, i.e., the notation may not support all capabilities of the external terminology schemata.
3) The notation used for expressing the mappings between the internal terminology schemata and the external terminology schemata (via the concept schema or not) must be capable of expressing detailed navigation and selection down to the value syntax of fields.

A.7 Requirements on notations for the distribution schemata

A.7.1 Introduction

A distribution schema defines one message class for communication between systems or with internal media of the application system.

The message class may be elementary, as in object-oriented communication, or the message class may be complex, as in file transfer.

The distribution schema states a view of the corresponding internal terminology schema.

A.7.2 Requirements

1) The notation for the distribution schema must be capable of expressing any view of the corresponding internal terminology schema.
2) The notation for the distribution schema must comply both to the internal terminology schema and the physical schema being used, but need not express a strict subset of either.
3) The distribution schema must be capable of defining any vertical or horizontal partitioning of data.
4) The distribution schema must be able to address its peer systems and media and define communication with these.

A.8 Requirements on notations for the physical schemata

A.8.1 Introduction

A physical schema defines the structure of the data as represented on an internal medium, e.g., on a hard-disk or a telecommunication line.

A.8.2 Requirements

1) A notation for a telecommunication line must be capable of defining the encoding of signals on this line.
2) There may not be a one-to-one mapping between signals defined in the encoding and messages defined in the distribution schema.
3) The notation for the telecommunication line must be capable of defining the mapping between message classes defined in the distribution schema and signal classes in the physical schema, and define the protocol for this implementation.
4) A notation for a database must be capable of defining the storage organization and storage details as perceived by the database designer.
5) The database views defined in the distribution schemata may deviate much from the physical organization of the database, defined in the physical schema.
6) The notation for defining the physical organization of the database must have means for stating proper references and access paths for its distribution schemata.

A.9 Requirements on notations for the system management schemata

A.9.1 Introduction

The system management schemata contain security and directory data for one or more systems.

The system management schema data are typically instantiated during execution only.

A.9.2 Requirements

1) A system management schema may contain data for more than one system, but must contain each system identifier and clearly indicate which resources, e.g., schemata and populations, relate to which system.
2) Security data may address application layer, contents layer and distribution layer data; seldom they will address layout and physical layers.
3) Directory data will provide overview of all layers and how they relate to each other.
4) Separate Recommendations apply for security and directory data.

Appendix I

Introduction to data architectures

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

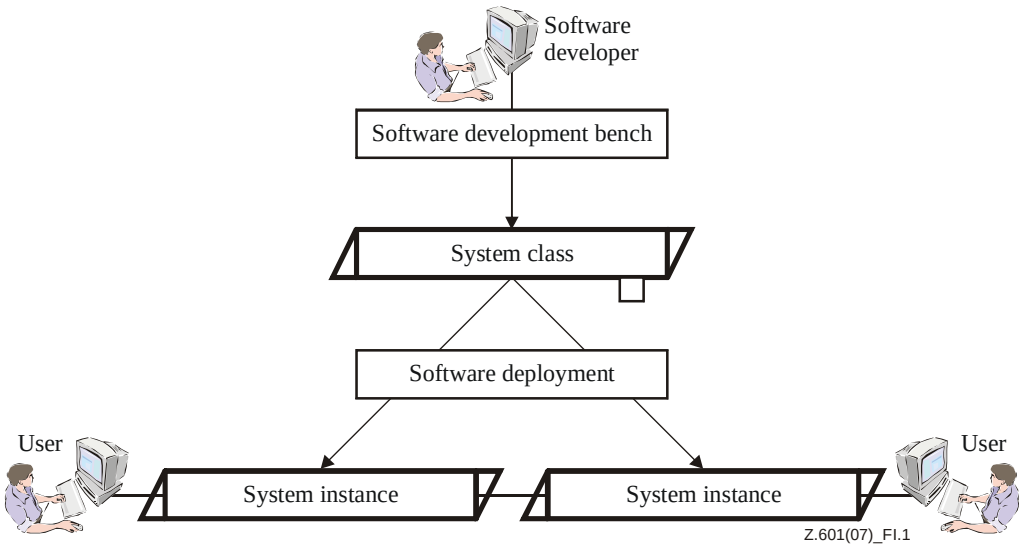
I.1 A system planning perspective

The clause provides an introduction to a terminology for systems planning of an organization.

A system instance is a set of data instances that are enforced as one consistent whole.

A system class is a set of data classes that prescribe constraints and derivations on data instances.

A system class is typically developed by a software development team. See the upper part of Figure I.1. A system class may be copied, shipped, distributed, configured, installed and instantiated into system instances to be used by the system users. See the lower part of Figure I.1.

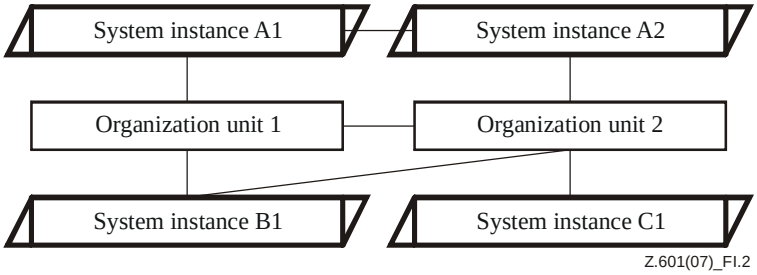


A recursive channel on the system class is instantiated into becoming a channel instance between the two system instances.

Figure I.1 – A system class is deployed into two system instances

A system class may be made up of software components. Software houses typically address the development of such components. Design of components and their interactions may be called portfolio management, and is not addressed in this Recommendation. However, the data architecture herein may be relevant to component design.

Rather than considering individual human users of systems, we may consider organizational users, i.e., organization units that interact with system instances. This is depicted in Figure I.2.



The two organizational users may be two operators using separate system instances of the same class A, one common instance of class B and a separate instance of class C.

Figure I.2 – Organizational users of system instances

Figure I.2 illustrates the core contents of a systems plan or a systems map of an organization. Even if communication channels between system instances are indicated, interoperation planning and interoperation maps are not addressed in this clause, as this clause only serves as an introduction to clause 6 on a data architecture of one system instance.

Configuration of system instances and their interactions is called integration. The integration should be based on target systems and interoperation plans, knowledge of software components to be used to implement the system instances and their interactions.

A transaction through a system instance causes an activation of that system instance. A transaction through an organizational unit causes an activity within this unit. The activity may involve activation of more than one system instance.

A transaction within a system instance is typically managed by the transaction handler of the software. A transaction of an organization unit is typically called a request, order, plan, trouble ticket or other, and its fulfilment and consistency are managed by the organization unit. An organization unit may comprise several subunits.

An order transaction may be a long transaction that spans several system instances and or organizational units. Also, new orders may be defined for each of these, and each unit or instance will have the responsibility to map back to the original order transaction.

The reader should note that we here have identified concrete system instances and organization units, their activations and activities, and have introduced no notion of abstract processes. The contents of a system is defined by the data it encompasses, not functionality or processes.

The data perspective will be kept during this Recommendation. This does not exclude combinations with a process perspective, but the systems may be designed by applying the data perspective only.

A way to conceptualise the data perspective is to imagine the entire system – be it a class or instance – to be made up of a set of statements in predicate calculus; Prolog may be considered to be a special case of this. The statements may be considered as data. And the statements need no notion of application dependent processes to be executed.

Note that as the term system is used with various meanings in the literature. The system notion of this Recommendation may be called one integrated system – instance or class.

I.2 A data perspective on a system

This clause introduces a data perspective on one system instance.

A system instance with its inputs, outputs, temporary and stored data is illustrated in Figure I.3.

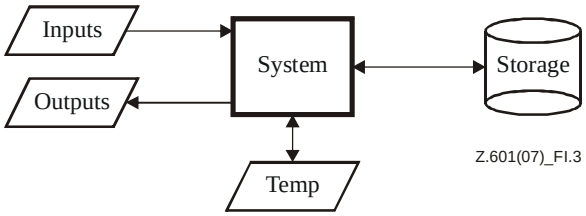


Figure I.3 – Depiction of a system instance

A processor (instance) may enforce the data classes on the data instances of a system instance. This is illustrated in Figure I.4.

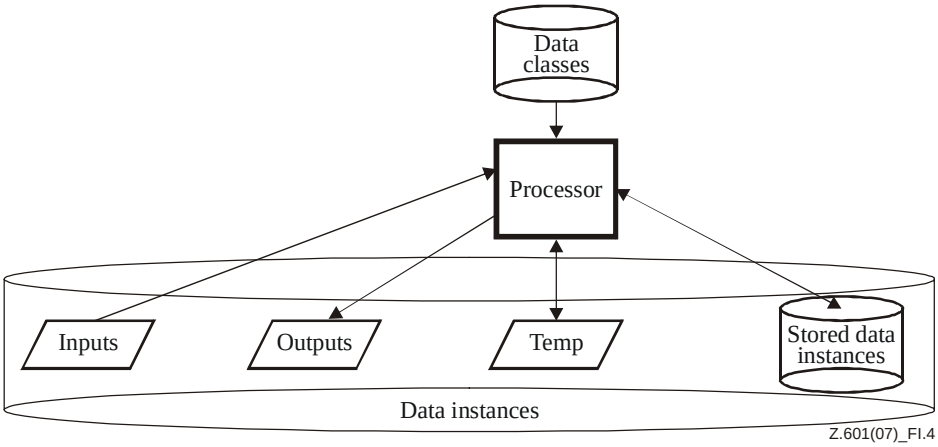


Figure I.4 – Data instances and classes of a system instance

Figure I.4 illustrates that the data classes provide classes for inputs, outputs, temporary and stored data. The system class defines classes for all these data. A system instance requires the data classes in order to operate appropriately, as the data classes prescribe the permissible structure and operations on their instance data.

A system instance may be executed on several interacting processors. This is called a distributed system instance. Note though that the collection of data on these processors only make up a system instance if consistency is enforced across all the data of the distributed system.

The consistency requirement means that the system ensures that no one can insert a statement p and its formal counterpart not-p simultaneously into the system instance. And nobody can retrieve both these statements simultaneously from the system instance. However, p may be stored in one system instance and not-p in another system instance, since there is no real time consistency enforcement across system instances.

Note that users may insert an informal statement p and its informal counterpart not p simultaneously into one system instance. The system instance only prohibits insertion of formally inconsistent statements. This prohibition is enforced during one run of a transaction.

Suppose the system instance uses strong typing, in the sense that a data instance must belong to only one class. Then input data and stored data may be permitted to be informally inconsistent within the system instance as long as a transaction is not defined to transform the input data to stored data in such a way that they become formally inconsistent.

The set of data classes of a system class is called a schema, and the set of data instances of a system instance is called a population. See Figure I.5.

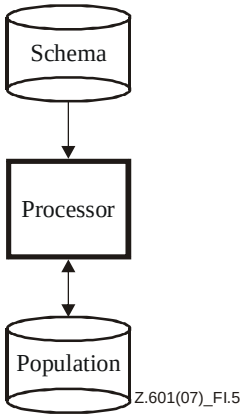


Figure I.5 – Schema and population of a system instance

However, data are not classes or instances in an absolute way; they are only so relative to each other. This is illustrated in Figure I.6.

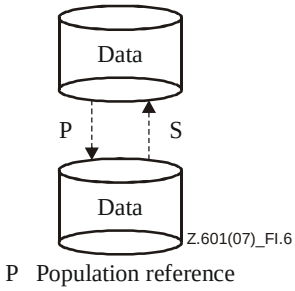


Figure I.6 – Schema and population references between data

Figure I.6 illustrates that schema-population references may be applied recursively. This will allow the data architecture of this Recommendation to be nested.

Note that the recursion indicated by Figure I.6 will only be possible if the data definition language used for the schema allows for recursion. For example, if the data language uses expressions like "managed object CLASS", this statement (syntax) can only be treated as a class and not as an instance. In order to allow for recursion of the data syntax, the notations should be designed such that data instances become pure copies of the data classes. This many-to-one mapping from instances to classes that maintain the structure of the data is called a homomorphism (on the syntax). See more on requirements for language design in Annex A.

Schemata and populations may be partitioned into subsets. Such subsets are identified in this Recommendation.

According to the definition, the schema defines all behaviour of a system instance. The behaviour may be defined as constraints on and derivation of data. These prescriptions on data may or may not be grouped into processes like order entry, order delivery, billing, invoice, etc. The perspective of

this Recommendation is that this kind of processes need not be identified within a system, as all behaviour may be prescribed as mappings between data, and a generic processor may enforce all kinds of constraints and derivations.

From the previous paragraph, we observe that a system class can be defined as a set of data classes, including their relationships and methods that express their behaviour. The process perspective on a system is not strictly needed.

I.3 Communication between systems

When communicating between system instances over a channel without loss of information, the receiving system instance receives the data set that is sent from the sending system instance. Here, the channel is not considered to be a system itself.

For the communicating system instances to be able to interpret the communicated data set correctly, they have to share common definitions for the communicated data set. These definitions are found in both system instances in a schema for the channel. These schemata must be identical for the data instances being communicated. This is what is called shared management knowledge in telecommunication management. See Figure I.7.

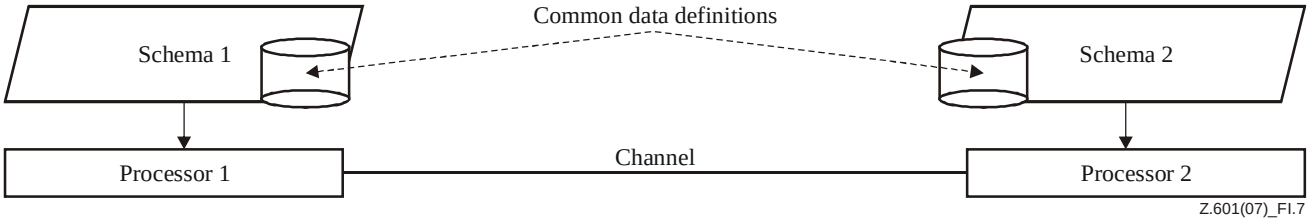


Figure I.7 – Common data definitions for the communicated data instances

Note that it is possible, and sometimes convenient, not to use the same data definitions in both ends of a data channel. Typically then the receiving system instance uses more generic definitions than the sending system instance. Hence, some of the classification information in the sent signal is lost. With knowledge of the sender's definitions, these definitions may sometimes be added later in the processing in order to provide a correct interpretation of the original data. This technique allows for using generic software for communication, while using more application specific software at the core of the system instances.

The data set that contains the communicated data instances is called a population. Note that only data instances that are sent together are enforced as a consistent whole and is called a population. Data which are not sent together, but are sent on different events, may be inconsistent with each other, and do therefore not make up a population. See Figure I.8.

The populations may share common schemata at both ends, but may be inconsistent.

Figure I.8 – Sent and received populations on a communication channel

An order instance is an example of a population, where consistency is enforced within the order, but not across orders. Even the order may be split into messages, where consistency is enforced within each message and maybe not across the entire order.

Each communicating system instance enforces its own consistency. Since the communicating system instances do not make up one consistent whole, they do not make up one joint system instance. Their data – both classes and instances – may be inconsistent, and the communicated data between the system instances may be inconsistent.

When using orders to convey data between systems, at any moment of time, the order may have affected one system instance, but not yet the other system instance. Only after some time, may the orders bring the system instances into consistent states with respect to this order instance. With respect to other order instances, the system instances may be inconsistent, as the updates have not yet taken place.

Orders are means to provide long transactions across several system instances and organization units. However, orders are not the only means to this end. As an alternative solution, one system instance may record states on the updating of other systems about certain data instances. This provides a simpler and not as complete solution to coordination between system instances, while orders provide identification of long transactions.

I.4 Communicating processes

Since each system instance enforces the constraints and derivations on data, and these derivations include mappings from input to output media of a system instance, there is no need to apply an application-dependent process perspective on communication between systems. The sending system, or the channel, need only to direct the data to the right sink system instance, and need not to know anything of its processing.

Also, there is no need of an application-dependent process perspective on the internal behaviour of a system instance. All its behaviour can be stated as constraints on data and transformation of data. Data are transformed within a system, not between systems.

I.5 Separation of media

Figure I.4 separates input and output data of a system instance. However, the output form of one dialogue step may become the input form of the next dialogue step. Hence, separation of input and output forms is not a very effective way of structuring data definitions on the input-output media.

Also, different media – such as a screen and a data communication line – may have very different characteristics. Therefore, it is convenient to separate the kind of media that provides interfaces to a system instance. We start with separating:

- external schemata, defining the external presentations to and manipulations by end users of the system instance, e.g., presentations on screens and reports;
- internal schemata, defining the internal organization and behaviour of data; the internal schemata define formats for storage of data in a database, for communication over telecommunication lines and other non-human interfaces.

This is illustrated in Figure I.9. Here, populations corresponding to the schemata are illustrated as well.

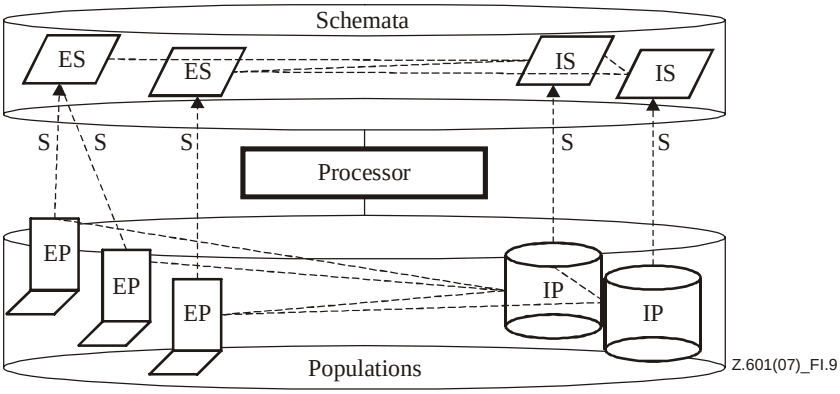


Figure I.9 – Example two-schema architecture

However, if we want to allow communication from all to all media, it may be very impractical to state the permissible mappings between every combination of any two schemata. The two-schema architecture may require up to (n+m)! mappings between schemata. The number of mappings may be reduced by using the three-schema architecture.

Appendix II

Comparison with other architectures

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

II.1 Comparison with ITU-T Rec. M.3020

[b-ITU-T M.3020] provides a methodology for specifying management interfaces between two physical systems.

This Recommendation provides a framework for the development of one system. This data architecture identifies candidate interfaces within one system as well as the interfaces on the boundary of this system. These interfaces at the boundary will be between systems.

[b-ITU-T M.3020] identifies specifications that may be related to the data architecture.

[b-ITU-T M.3020] is primarily aimed at the development of a set of Recommendations rather than of individual systems. However, [b-ITU-T M.3020] prescribes a requirements phase for management Recommendations. The data architecture prescribes no requirements capture, as it prescribes the specification of individual systems only, not their purpose relative to an organization.

However, this Recommendation focuses on specification of the external terminology and grammar as perceived by the end users. [b-ITU-T M.3020] focuses on specification of internal management interfaces, which may not be perceived by the end users.

[b-ITU-T M.3020] defines three phases and resulting outputs as follows:

– Requirements phase – Requirements.

– Conceptual design phase – Implementation independent specification.

– Implementation design phase – Technology specific specification.

In [b-ITU-T M.3020], the requirements for the problem being solved fall into two classes. The first class of requirements is referenced here as business requirements. The second class is referred to as specification requirements.

The specification requirements may include syntactical requirements to support end-user interaction at their human-computer interfaces. These syntactical requirements may need to be supported over any management interface. These syntactical requirements correspond to external terminology schemata of the data architecture as described in this Recommendation.

The output of the conceptual design phase will be an information model. This corresponds to a concept schema of the data architecture as described in this Recommendation.

The documentation from the implementation design phase will consist of two parts:

1) A technology-dependent data specification common for several interfaces, e.g., using GDMO or CORBA IDL, corresponding to an internal terminology schema according to the data architecture in this Recommendation.

2) A technology-dependent specification of each interface, e.g., using CMIP or CORBA IDL, corresponding to a distribution schema according to the data architecture in this Recommendation.

Bibliography

[b-ITU-T M.3020] ITU-T Recommendation M.3020 (2007), *TMN interface specification methodology*.

[b-ITU-T Z.352] ITU-T Recommendation Z.352 (1993), *Data oriented human-machine interface specification technique – Scope, approach and reference model*.

[b-Grietheusen] GRIETHEUSEN (J. J. van) (ed.): Concepts and terminology for the conceptual schema and the information base, *ISO/TC97, 1982 (ISO/TC97/SC5 – N695)*.

[b-Meisingset] Meisingset (A.): A data flow approach to interoperability, *Telektronikk 2/3.93*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2007