Information technology (IT) — Conceptual schema modelling facilities (CSMF)

Contents

List of figures

Foreword

ISO (International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Committee Drafts and Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75% approval by the member bodies voting.

International Standard ISO/IEC 14481 was prepared by Joint Technical Committee ISO/IEC/JTC 1, Information Technology.

This is the first edition of ISO/IEC 14481.

This International Standard contains the following annexes:

a) Annex A (informative) Bibliography

b) Annex B (informative) Mathematical Conventions and Notation

Introduction

This International Standard defines the applicable constructs that shall be contained in any modelling facility that is used to create a formal conceptual description of various aspects of an enterprise. The purpose of this International Standard is to provide a mechanism for end users and for information systems analysts, designers and constructors to communicate with each other in a formal way and to agree about contents of a conceptual schema (CS).

The organization of this International Standard is as follows:

a) Clause 1, Scope, specifies the scope and field of application of this International Standard.

b) Clause 2, Normative references, identifies additional standards that, through references in this International Standard constitute provisions of this International Standard.

c) Clause 3, Terms and definitions defines terms used in this International Standard.

d) Clause 4, Conventions and notation, introduces basic premises used in this International Standard.

e) Clause 5, Concepts for the CSMF, defines principles, concepts and an architecture which constitute the basis of the CSMF.

f) Clause 6, CSMF modelling foundation (defining schema), specifies the formal basis of the CSMF.

g) Clause 7, CS modelling constructs (normative schema), defines the set of constructs which comprise the CSMF.

h) Clause 8, Conformance, defines conformance requirements for the CSMF.

1 Scope

This International Standard defines the applicable constructs that shall be contained in a modelling facility that can be used to create a formal description of various aspects of all or part of any enterprise.

In this context a modelling facility is the basis of the implementation of a language and a tool to support the human endeavor of creating a formal description. The modelling facility defines the semantics of the language as a set of constructs and how they are related, though not the language syntax.

This International Standard uses the term enterprise to mean all or part of a commercial company, a government organization, an academic institution or some other social grouping such as a religious body, a trade union or an athletics federation. However, the modelling facility may equally be applied to a body of knowledge, the constitution of a country or of a federation of countries, or a set of laws.

A description prepared using a modelling facility conforming with this International Standard is a description which is, by definition, formal. It is formal in the sense that it conforms with the prescribed set of rules defined in this International Standard.

The capabilities in this International Standard are intended to make it possible to completely describe all relevant aspects of an enterprise. However, completeness of a CS can only be judged in terms of its purpose and depends on the skill of the user. One purpose intended for the modelling facility defined in this International Standard is for an analyst, designer, etc. to create a CS which is complete for the purpose of subsequent design and construction of a computerized information system to support the activities carried out in and by an enterprise. All considerations of how computerized information systems are to be constructed based on models using modelling facilities conforming to this International Standard are outside the scope of this International Standard. Another purpose is to support interchange and integration of descriptions of the enterprise.

The description which is prepared using this modelling facility is frequently referred to as a 'conceptual schema'. Hence the modelling facility is referred to as a 'conceptual schema modelling facility'.

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and the parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/TR9007:1987, *Information processing systems -- concepts and terminology for the conceptual schema and the information base*

3 Terms and definitions

For the purposes of this clause of the standard, every word used in this International Standard should be interpreted in the most general and appropriate sense given in the Oxford English Dictionary, unless defined in this International Standard. Words defined in this International Standard should be interpreted according to the definitions given in this document, or as defined in another International Standard and incorporated by reference.

3.1 Definitions

For the purposes of this International Standard, the following terms and their informal definitions apply. Some of these definitions rely on terms defined elsewhere in this International Standard.

3.1.1
conceptual schema
formal description of a UoD. A CS uses some normative formalism. It allows a formal description of entities contained in a UoD, along with properties and relationships between those entities. In addition it allows the description of formal rules, constraints, events, processes and other semantics. A CS may lead to one or more data models for information systems

3.1.2
conceptual schema language
formal language interpretable by either a computer or a human being, containing all linguistic constructs necessary to formulate the sentences that express the propositions in a CS

3.1.3
conceptual schema modelling facility
facility comprising semantic constructs that shall be contained in any CSL used to create a CS

3.1.4
data
representation forms of information dealt with by users or processed within an information system

3.1.5
external schema
definition of the external representation forms for the possible collections of sentences within the scope of a particular user's view, including the manipulation aspects of these forms

3.1.6
information
any kind of knowledge about things, facts, concepts, etc., of a UoD that is interpretable to and exchangeable among users

3.1.8
internal schema
definition of the internal representation forms for the possible collections of sentences processed within an information system

3.1.9
mapping
ability to associate one construct in one syntactic form with an equivalent, or partially equivalent, construct in another, alternative syntactic form

3.1.10
universe of discourse
a collection of entities of interest

3.1.11
user
anybody or anything that interacts with an information system

3.2 Abbreviations

For the purposes of this International Standard, the following abbreviations apply:

CS conceptual schema
CSL conceptual schema language
CSMF conceptual schema modelling facility
UoD universe of discourse

4 Conventions and notation

4.1 Conventions

The following informative elements may be used in the definition of the CS modelling foundation and constructs.

4.1.1 Notes

In several places in the body of this International Standard, informative notes appear. For example:

NOTE This is a note.

Notes do not belong to the normative part of this International Standard and conformance to material specified in notes shall not be claimed.

4.1.2 Examples

In several places in the body of this International Standard, informative examples appear. For example:

EXAMPLE This is an example.

Examples do not belong to the normative part of this International Standard and conformance to material specified in examples shall not be claimed.

4.2 Notation

The formal definitions of the CS modelling constructs in 7. use typed predicate logic. For example, in the definition of event, the part of the definition (∀e:event) is read as 'for every event e'. This is more concise than the untyped version. The formal definitions make use of a number of logical symbols. The readings given to them in this International Standard are listed below.

logic	reading
∃x	there exists an x
∃!x	there exists exactly one x
∀x	for all x, or for every x
x : y	x is of type y
x∈y	x is an element of y
x⊃y	x is a super class of y
x⊂y	x is a subclass of y
x..y	from x to y (denotes a range)
<x,y,z>	sequence x, y, z
x ∧ y	x and y
x ∨ y	x or y
~x	not x
r(x)	r is true of x
x ⇒ y	x implies y
x ⇔ y	x if and only if y (alternatively, x iff y; x equivalent y)
◊x	possible x
[]x	necessary x

tv(x) x has a truth value (e.g., true, false, unknown)

The usual mathematical combinations of these symbols are described in an informative annex (see annex D).

5 Concepts for the CSMF

5.1 Principles

The CSMF shall adhere to the following principles.

5.1.1 "100% principle"

The CSMF enables the production of CSs that obey the "100% Principle" of ISO/TR9007 - All relevant structural and behavioral rules, laws, etc. about the UoD must be described in a CS. That CS defines the UoD.

5.1.2 "Conceptualization principle"

The CSMF enables the production of CSs that obey the "Conceptualization Principle" of ISO/TR9007 - A CS should only include conceptually relevant aspects, both structural and behavioral, of the UoD. All aspects of external or internal data representation are to be excluded. In particular it enables the production of a CS which is independent with respect to physical implementation technologies and platforms.

5.1.3 "Helsinki principle"

The CSMF enables the production of CSs that obey the "Helsinki Principle" of ISO/TR9007 - Any meaningful exchange of utterances depends upon the prior existence of an agreed upon set of semantic and syntactic rules. The recipients of the utterances must use only these rules to interpret the received utterances, if it is to mean the same as that which was meant by the utterer.

5.1.4 Distinction of a concept and its representation

The CSMF allows distinction between the concept and the representation of the concept.

5.1.5 Concrete CSL syntaxes

The CSMF constructs enable mapping to and from concrete CSL syntaxes.

5.1.6 Nature of the world

The CSMF makes minimal assumptions concerning the nature of the world.

5.1.7 Extensibility

The CSMF provides mechanisms for extending the standard set of constructs.

5.1.8 Self description

The CSMF allows the CS modelling constructs defined in this International Standard to be self-described.

5.2 Basic concepts for the CSMF

The basic concepts underlying this International Standard are as follows:

5.2.1 CSMF-conformant implementation

A CSMF-conformant implementation has a set of syntactic rules which define well-formed statements in a CSL.

5.2.2 Operations in a CSMF-conformant implementation

The CSMF-conformant implementation has a set of operations which can be applied to statements in the CSMF-conformant implementation to create new statements.

5.2.3 CSMF semantic interpretation

The CSMF semantic interpretation is a set of rules for interpreting statements in a CSMF-conformant implementation in terms of a UoD. In particular, the rules ensure the correspondence of terms with entities, the assertion of statements as being true of that UoD, the preservation of truth through CSMF operations in a CSMF-conformant implementation, and the consistency of sets of statements in a CSMF-conformant implementation.

5.3 Concepts of use

A CSMF is targeted directly at developing conceptual models and using them to support the exchange of information among different users and their supporting information systems. CSMF, like other modelling methods and tools, will most likely be implemented as part of broader IT environments, including for example repository technologies, CASE tools, and communications protocols. These environments may require the joint application of several industry standards. Within these implementation environments, CSMF may be used directly to produce CSs, or they may be used indirectly, in the sense of interrelating and integrating models produced by more focused, specialized enterprise modelling methods and tools. CSMF may also be used in mixed mode, i.e., drawing some portion of their CSs from the conceptual aspects of enterprise models produced by other methods and tools, but adding further semantic details about the interrelationships among partial models. An example might be relating the information in an information model produced using an entity-attribute-relationship (E-A-R) method or tool to the processes in a process model produced using Petri Nets, and then representing business rules about what information can be used in which processes under what circumstances and to support which decisions.

The basic concepts and the principles of the preceding sub-clauses of this CSMF standard provide that a CSMF implementation can be used in the following ways:

5.3.1 Enterprise modelling

A product providing a CSMF output interface can be used to create and maintain a CS of any enterprise or other UoD. The interface provided for the modellers is at the discretion of the product provider. It may use the same modelling concepts and constructs as the CSMF semantic interpretation or any other modelling constructs for which the product provides a mapping to the CSMF constructs.

5.3.2 Use for application development

A CS can be output from a CSMF for use in the process of building an information system. This use can be wholly automated, partly automated, or wholly subject to human transcription. Applications can then be built according to the CS specification.

5.3.3 Configuration management and traceability

Configuration management of CS models deals with knowing which rules are described in which version of a CS. This may include the specification of rules that support a specific application area.

Configuration management in a CSMF-conformant implementation provides traceability from an entity to all the rules in a CS that apply to that entity. Traceability enables identification of the rules that support a specific application area.

5.3.4 Display of semantic content

Where an information system is based on a CS, this CS provides a basis for interpretation of any output generated by the information system. The CS may be used in human-readable form or via a CSMF-conformant implementation.

The CS may be used in this way by any person requiring understanding of the output of the information system, whether from the organization owning the CS and information system or any other organization given access to them.

5.3.5 Interchange of CSs

Should a CS of a particular UoD be wholly or partly created using one CSMF-conformant implementation, it may be output from that and input to a different CSMF-conformant implementation providing a CSMF input interface. The second conformant CSMF implementation can be used to examine, change or add to the CS through whatever user interface it provides, whether the same or different from the first CSMF-conformant implementation.

The property of interoperability allows an information system to interchange and interpret the syntax (form) and semantics (meaning) of data or processes with one or more other information systems which also conform to the standards of interoperability. Such systems may be centralized (local) or distributed. It is possible for one information system on a computer system to be interoperable while other information systems on the same computer system are not. A CSMF-conformant implementation supports interoperability by providing a common basis for interchanging and interpreting the syntax and semantics of enterprise data.

Three different degrees of interoperability can be recognized:

a) interchange among heterogeneous CSMF-conformant implementations on a purely syntactical basis. This involves little or no semantic depth.

b) interchange of semantics among heterogeneous CSMF-conformant implementations, but where the semantics are imbedded, not explicit. This involves some expansion to semantic depth.

c) interchange and interpretation of semantics among heterogeneous CSMF-conformant implementations. The semantics of the CSs are being integrated rather than just interchanged. The semantics are explicitly defined as well as using standard syntax. This allows for interpretation.

This International Standard is targeted primarily at c).

5.3.6 Reconciliation of CSs

Where two or more CSs have been created independently but their UoDs may overlap to a greater or lesser extent, a CSMF-conformant implementation providing a CSMF input interface enables access to the two CSs. This facilitates recognition of similarities and differences between the two CSs. Where the CSMF-conformant implementation provides checking of consistency between sets of CSMF syntax statements, this aids the process of reconciliation.

5.4 Concepts to be modelled

This sub-clause defines concepts required to model an enterprise and which require representation in a CS.

5.4.1 Information concepts/enterprise information

Enterprise information is data that is used in the enterprise by the various activities which are performed in the enterprise. Enterprise information is not limited to that which is handled in a computerized system.

5.4.2 Process concepts/enterprise activities

A fundamental aspect of modelling an enterprise is modelling enterprise activities (sometimes called enterprise processes or enterprise functions). These activities may be specified at various levels of detail as a process hierarchy. This should be distinguished from the application or data manipulation processes. The latter are specified in application programs or using a data modelling facility.

The CSMF provides facilities to specify the users' view of the processes which the information system can perform. These may be referred to as user tasks. The concept of a user task is positioned between the concept of an enterprise activity and the concept of an application process. The positioning is not a matter of abstraction level or of complexity. A user task belongs at the interface between the enterprise activities and the computerized information system.

This International Standard makes it possible for a user task to be designated in such a way that it is meaningful to an enterprise expert or to a user of the computerized information system, in the sense that a user is able to point to it from among a selection containing several such tasks. How a user task is initiated is regarded as an implementation issue and not appropriate for this CSMF. Which user tasks may be initiated is part of a CSMF when it is used in prescriptive mode, but not when it is used in analytic mode.

This CSMF provides for the representation of enterprise events (as distinct from enterprise activities).

5.4.3 Temporal considerations

This CSMF provides for the representation of temporal aspects of the enterprise.

a) definition of temporal constraints between enterprise activities;

b) definition of temporal constraints between enterprise events;

c) definition of time zones associated with locations at which enterprise activities are performed and enterprise events happen.

5.4.4 Business rules

Enterprise rules are rules which exist about the enterprise and which determine the way in which the enterprise conducts its business. Some enterprise rules may have been formalized in procedure manuals. Some enterprise rules may be determined by national or international legislation. Other enterprise rules may be simply no more than accepted practice.

5.5 Positioning the CSMF

The CSMF is positioned in three ways. The first way is relative to existing modelling techniques. How the CSMF relates to existing modelling technique is fundamental to understanding the positioning of the CSMF.

The second and third ways of positioning the CSMF are related to each other. The second is called processor architecture and the third is called a schema architecture. The relationship between the two architectures is explained as part of this clause.

It is important to the discussion of the schema architecture and the processor architecture to appreciate the difference between the following items:

a) The way in which a modelling technique itself is defined;

b) The way a modelling technique is used to define a model.

Item a) is the problem faced by the designers of a modelling technique and by the authors of this International Standard.

Considering item b), if the modelling technique is something for creating a CS, then the "model" referred to in b) is a CS, and b) refers to the way in which a CS is defined. The way in which a CS is defined refers to a CSMF syntax and possibly also to the associated CSMF operations.

5.6 Relationship to existing modelling techniques

5.6.1 Existing modelling techniques

There are many modelling techniques in use for handling some part of one or more of the concepts listed in 5.4. Some of these modelling techniques are already the subject of standards and some are under consideration for standardization. This clause indicates how the CSMF relates to such modelling techniques. The term existing modelling technique (EMT) is used in this annex to refer to any such modelling technique.

NOTE An EMT can also be thought of as belonging to a family (such as the family of entity relationship modelling techniques). For this reason, the concept of a generic modelling technique (GMT) is introduced in this International Standard. It is not necessary for an EMT to belong to a GMT. For a GMT to which several EMTs are related, each EMT in the family may have its own syntax. However, the association between each EMT and a GMT is established in terms of the semantics of the constructs in each.

5.6.2 Role of constructs in modelling techniques

A modelling technique can be regarded as using a number of inter-related constructs. Each use of the technique creates a number of different instances of each of these constructs.

The constructs are used for the purposes of modelling an enterprise. The constructs are not normally those concepts which are found to be part of the enterprise being modelled — except when a modelling technique is used to model itself or some other modelling technique.

The task of defining a modelling technique reduces to that of identifying which constructs are pertinent to the modelling technique and how these are inter-related.

When an EMT is used, its constructs will typically be a subset of the CS modelling constructs.

5.6.3 Role of syntax in modelling techniques

In order to make use of an implementation of an EMT, it shall be realized using a syntax. The syntax may take the form of a diagramming technique or it may take the form of a structured language based on a prescribed alphabet (such as is used in natural language). It is not the aim of this International Standard to standardize any such syntax although that may be done in other standards.

The CSMF as defined in this International Standard is defined in terms of a set of constructs which in this International Standard are referred to as the CS modelling constructs. This International Standard does not define a

syntax for the use of the CSMF or for any other modelling technique. The relationship between the CSMF and an EMT may be defined in terms of the semantics of the constructs in each.

5.6.4 Concepts excluded from the CS modelling constructs

The set of CS modelling constructs is chosen to be any of those pertinent to the task of modelling an enterprise. The following concepts are explicitly excluded from the set of CS modelling constructs.

a) Concepts specific to the technical (including performance-related) aspects of constructing a computerized information system (for example index and assignment of data to storage);

b) Concepts specific to an approach to the prescriptive design of the computerized information system as opposed to the analysis of the enterprise (for example object-oriented programming language related constructs).

A CS is not regarded as a container for all specifications that precede the design of a computer information system. For example, issues relating to the evaluation of alternatives such as a tailor-made system versus a ready-made package, or issues relating to risk assessment, would not normally be part of a CS.

Any such concepts in an EMT should not be used in building a CS.

5.6.5 Role of CS modelling constructs and services

A set of constructs for the CSMF is defined in this International Standard. To avoid confusion with other constructs, these are referred to in this International Standard as the CS modelling constructs. To be able to make use of the facilities provided by a CSMF, a set of services is defined in this International Standard. These services are defined formally, but the precise syntax of the services is not standardized.

5.6.6 Relationship of CSMF to RM - ODP

This International Standard specifies a standard for a CSMF. A CSMF permits specification in a CS of concepts and terms about one or more UoDs, also referred to as subject area(s) or enterprise(s). A CS describes or models a UoD; it does not describe anything other than the UoD. In some cases, the UoD being modelled may be an information system, although this is not often the case. Even in such cases, the information system is being modelled as a UoD in itself, not for purposes of specifying the information system requirements, design or implementation as part of a systems development life cycle.

The Reference Model - Open Distributed Processing (RM - ODP) takes the position that in order to develop an information system for all or part of a business enterprise, the relevant parts of that enterprise have to be specified in an abstract (i.e., implementation-independent) and precise manner (see ISO/IEC 10746-1). As part of this process, the specification of the purpose, scope and policies of the enterprise is the subject matter of the enterprise viewpoint of RM - ODP. This RM - ODP specification is concerned with an abstract view of the enterprise, and therefore this aspect of the ODP standard and the CSMF standard share a common focus. However, the ODP approach stipulates that all specifications developed should be based on the same fundamental concepts as specifications of any other ODP systems, including information management systems. This specifically requires use of the concepts contained in ISO/IEC 10746-2. Some of those concepts are the same as, or similar to, concepts contained in this International Standard, or are derivable from the concepts contained in this International Standard. For this reason, the enterprise viewpoint language of RM - ODP may be considered as a specialized CSL for some modelling purposes. In particular, the enterprise viewpoint language may be considered as a starting point for a complete system specification (in particular, an information system specification). In the case of the other four basic ODP viewpoint languages, the specifications produced are specifications of an ODP system or application with an emphasis on details different from the details of the purpose, scope and policies of the enterprise. Such specifications may be considered as extending the RM - ODP enterprise viewpoint specification to a level sufficient

to specify the other concerns of the (business or information) system, specifically, the information, computational, engineering and technology ones.

5.7 Processor architecture

5.7.1 Representation of processors

In this clause, the diagramming notation of the Reference Model of Data Management (RMDM) is used (see ISO/IEC 10032). This is shown in figure 1.

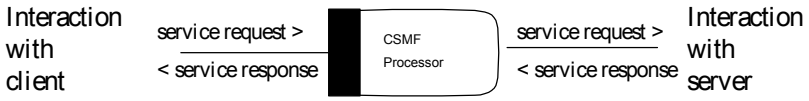

Figure 1 — Notation to represent a processor and its interactions

Figure 2 — Single processor view of CSMF

5.7.2 CSMF processor

Using the concepts of the RMDM, a user perception of the provision of services for the creation and maintenance of a CS may be expressed by a single processor, as shown in figure 2.

Figure 3 — User perception of CS processing

The CSMF processor in figure 3 provides services to its users in terms of a CSL for some specific modelling technique with its own presentation syntax. This interface is outside the scope of this International Standard. The user perceives access to the CS via the CSMF processor.

5.7.3 CSMF semantic processor

This simple view of a single processor may be decomposed to show further detail, as shown in figure 4.

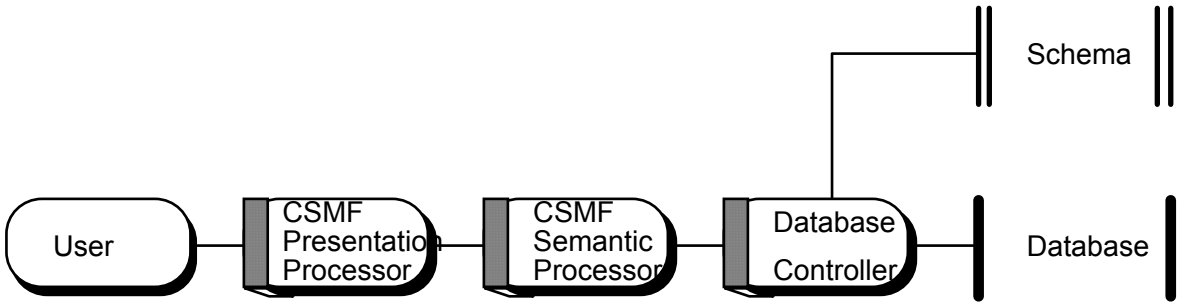

Figure 4 — CSMF semantic processor

In figure 4, the CSMF presentation processor provides the same services to a user as the CSMF processor in figure 3. The processing for these services makes use of services provided by the CSMF semantic processor, which are expressed in terms defined by this International Standard. The services at this interface to the CSMF semantic processor reference the CS modelling constructs and use these to express a CS.

The services at the interface to the database controller are used for storage of a CS. An implementation of this International Standard may make use of database management services, for example a database management system based on SQL (see ISO/IEC 9075) to store a CS in a database. The definition of this database (i.e., its schema) is outside the scope of this International Standard.

5.7.4 Export/import processor

Transfer of a CS between CSMF implementations (i.e. export and import) is defined in terms of the data available via the services of the CSMF semantic processor. The processor for this purpose is shown in figure 5, which also enables E/I services to be invoked via the CSMF presentation processor, i.e., in a way appropriate to the user interface provided by this processor.

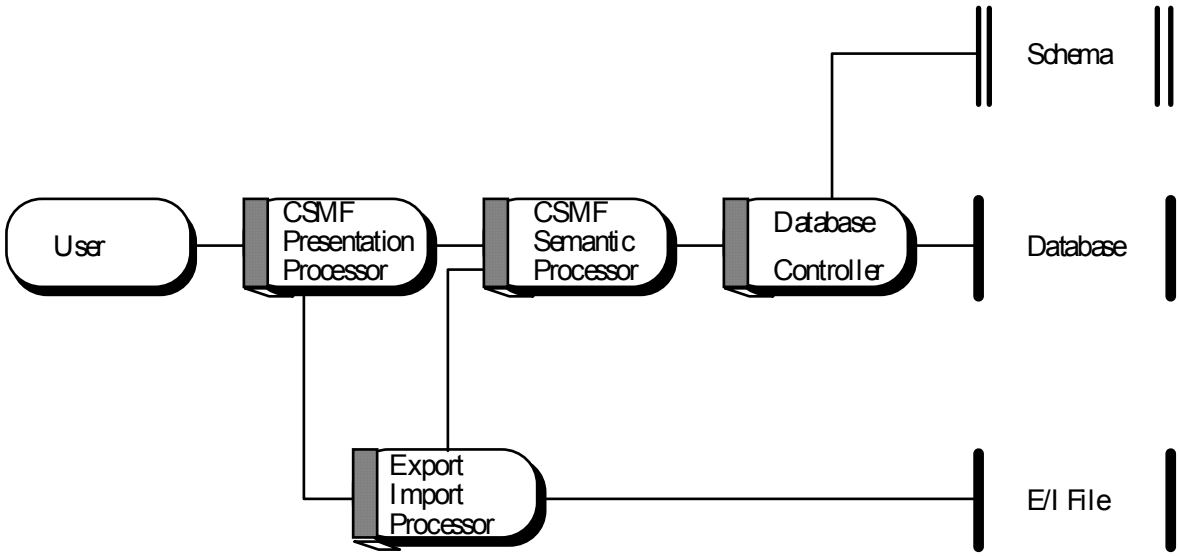

Figure 5 — Export/import of a CS

5.8 Schema architecture

The schema architecture is depicted in figure 6. The schema architecture described herein is comprised of four distinct kinds of schemas (defining schema, normative schema, meta-models/modelling schemas, and application schemas) situated in three partitions called application models, meta-models and meta-meta-models. This figure represents a logical architecture for organizing information described in the form of a model/schema.

The schema architecture can, alternatively, be envisioned as an "onion skin" (see ISO/TR9007). In that metaphor, the innermost layer (the meta-meta-model) contains the defining schema. The CS modelling constructs, which are part of the meta-model partition, are described in a language based on the fundamental concepts in the defining schema. The meta-models (modelling schemas), which are also part of the meta-model partition, conform (in varying degrees) to the CS modelling constructs defined in the normative schema. The normative schema can also be called the 'root' meta-model. Application schemas represent the outermost layer are specified using one of the modelling schemas. 6. addresses the defining schema component of the CSMF architecture; 7. addresses the normative schema component.

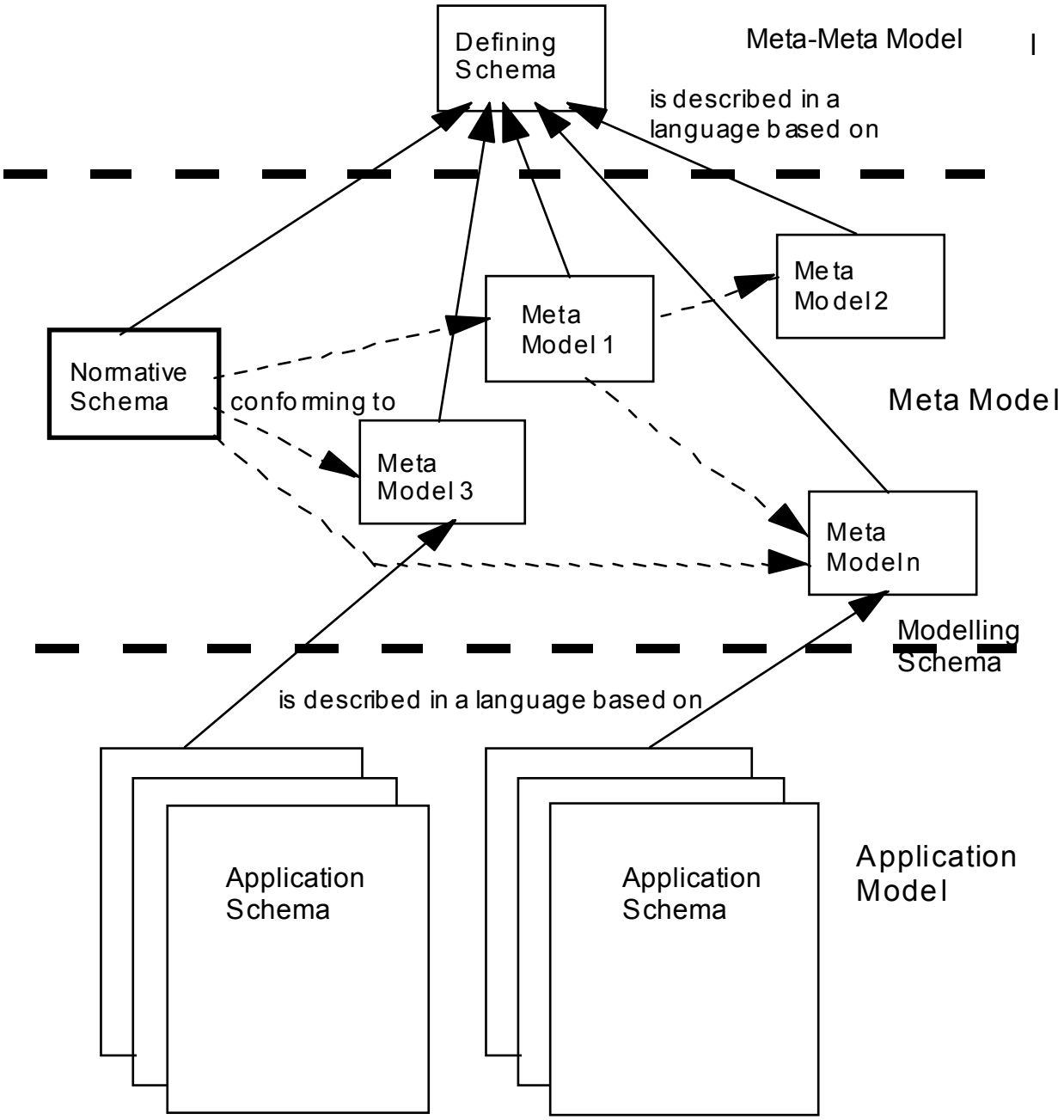

Figure 6 — Schema architecture

5.8.1 Dimensions of the CSMF architecture

Given the range of intended uses for CSs, the logical architecture for CSMF encompasses multiple dimensions. These are in addition to the concrete processor architecture described earlier in this clause. These dimensions, which form a taxonomy of schemas, include the following architectural elements.

5.8.1.1 Types of model expressiveness

Schemas may contain information of a structural, behavioral, and modal nature.

5.8.1.2 Three-schema architecture

Representations may be organized into at least the following three schemas (see ISO/TR9007):

a) an external schema (ES) for presentation and manipulation by an application,
b) a conceptual schema (CS) for meaning, and
c) an internal schema (IS) for physical storage.

5.8.1.3 Life-cycle phases

Information may be described and controlled throughout the systems engineering life-cycle. This means that a CSMF should be tightly coupled to various software engineering tools such as CASE tools, repositories/information resource dictionaries, etc.

5.8.1.4 Elaboration of the schemas in the schema architecture

There are three basic partitions in the schema architecture of the CSMF: application model, meta-model, and meta-meta-model. These are in addition to the information base, which lies outside of the CS. The information base and the three partitions comprising the schema architecture may be described in more detail as follows (see also figure 6).

5.8.1.4.1 Information base

Instances of data and objects are manipulated by the processes supporting business processes in the UoD using manual and computerized information systems. The collection of instances for a particular CS is known as the information base for that CS.

5.8.1.4.2 Application models: application schemas

An application model or application schema defines the types of objects, data, and processing that are instantiated in the information base.

An application schema is typically expressed using the syntax and semantics of one (or more) modelling language(s) represented in a meta-model. An application schema may consist of a number of schemas, each of which may be described in a different language based on a meta-model. In some cases, however, such application schemas may be described by using one single modelling language.

These schemas contain the individual objects involved in the software engineering process of specifying and implementing an information system to support the business processes in a UoD within one (or more) enterprise(s). The kinds of definitions that can be included in an application schema, the processes for creating and manipulating those definitions and the CASE (Computer Aided Software Engineering) tools supporting these processes are defined in a meta-model supported by a corresponding concrete CSL.

5.8.1.4.3 Meta-models: modelling schemas and the normative schema

A meta-model or modelling schema contains the definitions of the concepts, terminology, operations and assumptions needed to construct application schemas. In other words, meta-model descriptions contain the syntax and semantics of various modelling or representation languages, schemes or paradigms used for modelling within, for example, the context of the software development life-cycle. They conform (in varying degrees) to the defined CS modelling constructs in the normative schema (also called the "root meta-model").

Meta-models may either conform directly to the CS modelling constructs defined in the normative schema or to the constructs defined in another meta-model that conforms to the CS modelling constructs.

They may also be described using the constructs and concepts of the defining schema.

The normative schema defines the concepts, terminology, operations and assumptions needed to create conceptual schemas/models. The CS modelling constructs in the normative schema are defined based on the fundamental descriptive concepts in the defining schema. Every meta-model will utilize a particular concrete CSL defining a specific representation, conforming to the normative schema.

5.8.1.4.4 Meta meta-model: defining schema

The defining schema defines the concepts, terminology, operations and assumptions needed to be able to specify the CS modelling constructs in the normative schema. It is itself expressed in natural language (e.g., English or French), with enrichments from mathematics, set theory, ontology, phenomenology, philosophy, logic, linguistics etc. Since it naturally contains some base constructs whose definitions are archetypal, it is defined less formally than the normative schema.

The defining schema is not considered as having a direct operational role in relation to the meta-models, and is not the primary subject of this International Standard. Its role is to provide a means for enabling the definition of the concepts in the normative schema and for evaluating proposed changes to the normative schema.

Since the normative schema will contain constructs as rich as first-order predicate logic, the defining schema must itself have constructs that provide the functional capability of first-order predicate logic in order to enable a full description of the syntax and semantics of the normative schema. The defining schema expresses them in the syntax of (enriched) natural language

5.8.2 Relationship of schema architecture to ISO IRDS services interface levels

The schema architecture partitioning in application schema, modelling schema, normative schema and defining schema are not intended to represent the same notion as the concept of level pairs in the Information Resource Dictionary System (IRDS) services interface (see ISO/IEC 10728). The purpose of the level pair concept is to describe specifically the relationship between the lower level of one level pair and the upper level of the next lowest level pair. The purpose of the CS architecture is to position the four major schemas with respect to one another based purely on the nature of the schemas and their relationships.

The CSMF schema architecture can, however, be compared with the IRDS services interface as follows:

a) application model partition:
   1) the information base is the lower level of the IRDS application level pair
   2) the application schema partition is the upper level of the application level pair
   3) the CSL of a meta-model utilized to produce an application schema is the lower level of the IRDS dictionary level pair

b) meta-model partition:
   1) modelling schemas are the upper level of the IRDS dictionary level pair
   2) the normative schema (the "root meta-model") is the upper level of the dictionary level pair

c) meta meta-model partition:
   1) the defining schema is the lower level of the IRDS definition level pair

The important fact to note is that in these architectures, the normative schema of this International Standard and the definition of the specific meta-models can be positioned in the same architectural aspect relative to each other in the same way -- as indeed can SQL schemas in IRDS.

To clarify this, it is useful to recognize there is a dimension of progression along the IRDS dictionary level pair which is quite orthogonal to the dimension "down the levels." The defining schema is therefore playing exactly the same role as the IRD definition schema in the IRDS services interface.

In a particular CS meta-model, a representation of each of the meta-models could be physically stored on the lower level of the definition level pair. When these representations are "activated" (using the terminology of the IRDS services interface and the reference model of data management [see ISO/IEC 10032]), physical schemas for each would be "created" (per the upper level of the dictionary level pair) and application-specific schemas could be developed (at the lower level of the dictionary level pair).

6 CSMF modelling foundation (defining schema)

This clause describes the fundamentals of the CSMF modelling foundation and provides short, precise, although "intentionally informal" definitions of the concepts and terms. Wherever possible these definitions conform with the meaning most closely associated with natural language.

It is important to differentiate between the language used to write a CS and the concepts described within that language. This clause describes the underlying concepts for the CS modelling constructs defined in 7.

6.1 Fundamental concepts for the UoD

6.1.1
entity
any concrete or abstract thing of interest

NOTE Entity is a universal concept that includes static and dynamic things, associations among the things, and any thing perceived to exist in some real or hypothetical world.

6.1.2
universe of discourse
all those entities of interest that have been, are, or ever might be in a selected portion of a real or hypothetical world.

6.1.3
proposition
a conceivable state of affairs concerning entities about which it is possible to assert or deny that such a state of affairs holds for those entities

NOTE 1 A proposition can concern one entity, several individual entities, groups of entities, etc.

NOTE 2 The entities in the UoD may be concerned with many states of affairs. A proposition is a predication of a conceivable state of affairs about one or more entities.

NOTE 3 A proposition is an abstract entity which serves as a denotation of one or more sentence(s) in one or more CSL(s). A proposition has a truth value.

NOTE 4 In practice the distinction is often made between propositions about the actual state of individual entities, and, propositions about which behavior of entities may or may not be permissible or possible. The words "rule" and "constraint" refer in particular to propositions of this latter kind.

6.1.4
property
a proposition concerning only one entity is called a property of this entity

6.1.5
semantic predicate
the component of a proposition that associates the one or more entities that form the proposition

NOTE A semantic predicate does not entail any means of linguistic expression.

6.1.6
entity state

the set of propositions in which that entity participates and that are true in the UoD at a given time

NOTE For the propositions to be true, the entity state includes the existence of the entities referenced in the propositions in the set.

6.1.7
UoD state
the set of all propositions that are true in the UoD at a given time

NOTE 1 For the propositions to be true, the UoD state includes the existence of the entities referenced in the propositions in the set.

NOTE 2 The UoD as a whole can be seen as the set of all possible UoD states.

6.1.8
sequence
a sequence of length n $<x_1, \ldots, x_n>$ is a function from the integers 1 through n where each x_i is an entity in the sequence

6.1.9
successor
a state or event x_2 is a successor of state or event x_1 if the time of x_1 precedes x_2, or if x_1 precedes x_2 in some sequence

6.1.10
immediate successor
a state or event x_2 is an immediate successor of state or event x_1 if x_2 is a successor of x_1 and there is no state or event x such that x is a successor of x_1 and x_2 is a successor of x

6.2 Fundamental concepts for representation of the UoD

6.2.1
sentence
a grammatically-allowable construct in a language such that the construct is used to express a proposition

NOTE 1 Communication about the UoD can take place by describing, through sentences, propositions which hold for the entities in their states of affairs.

NOTE 2 Linguistic constructs may be considered entities.

NOTE 3 A sentence s is about an entity x if s contains a sub-expression whose denotation includes x.

6.2.2
denotation
a function from an expression in a CSL to an entities

6.2.3
term
a syntactic construct that refers to a entity

NOTE Sentences consist of terms contained in a grammatical structure.

6.2.4
linguistic predicate
the component of a sentence that associates the one or more terms that form the sentence

NOTE A linguistic predicate is purely syntactic.

6.3 Class, type and instance

6.3.1
class (of entities)
a possible collection of similar entities to which a semantic predicate may be applied to form a true proposition

NOTE 1 A proposition may hold for a collection of similar entities. This collection is called a class.

NOTE 2 Each class of entities is determined exactly by its possible members. Any particular entity may be a member of many classes and the set of classes of which it is a member may change.

NOTE 3 The semantic predicate that determines the class might describe a state of affairs of arbitrary complexity.

6.3.2
type (of a entity)
a semantic predicate that determines a class that has been selected for the purpose of modelling

6.3.3
instance (of a type)
occurrence (of a type)
a particular entity, for which the designated type proposition holds

NOTE 1 The notion of instance or occurrence is usually associated with the notion of type.

NOTE 2 The notions of class and type are used to establish collections of necessary propositions. A collection of relevant necessary propositions may be identified with a specific type, that hold for all possible entities of this type.

NOTE 3 The CS modelling constructs defined in 7. implicitly use the type-instance notion.

7 CS modelling constructs (normative schema)

It is important to differentiate between the language used to write a CS and the concepts described within that language. This clause presents the modelling concepts defined in this International Standard. Each is described with a name, a natural language definition, a formalized restatement of the natural language definition, rules and examples.

The CS modelling constructs describe concepts. Such concepts underlie the things in themselves as they exist in some real or imagined world, our conceptual representations or models of those things, and the languages (spoken words, written text, graphics, etc.) that are used to symbolize and manipulate them.

Each CS modelling construct is identified in this International Standard by a specific term.

This normative clause defines a deliberately limited number of modelling constructs that collectively form a complete basis for defining a CSL and its semantics. Many of the modelling constructs are general in the sense that for any particular CSL, multiple constructs may be utilized to address a given CSMF construct.

The definition of each modelling construct in the set applies to both types and instances of the construct.

7.1
UoD entity
an entity that is recognized as having real or hypothetical existence in the UoD

FORMALIZED NATURAL LANGUAGE DEFINITION
($\forall$x) UoD entity (x)

RULE Every UoD entity must be distinct.

TYPE EXAMPLE 1	hotel
TYPE EXAMPLE 2	person
TYPE EXAMPLE 3	sales-order
INSTANCE EXAMPLE 1	Wawona Hotel
INSTANCE EXAMPLE 2	R. Hotaka
INSTANCE EXAMPLE 3	54365

NOTE 1 An UoD entity may be both a type and an instance (of another UoD entity), depending on the UoD being modelled..

NOTE 2 Whereas entity (see 6.1.1) is a general concept, UoD entity is a specific concept which is intended to model static, structural and independent things rather than dynamic things or associations among things.

NOTE 3 The concept of state, as used for example in Petri nets and state transition diagrams, may be modelled using UoD entities.

7.2
fact
a proposition involving one or more entities each of which plays a distinct role, and which is held to be meaningful and true

FORMALIZED NATURAL LANGUAGE DEFINITION
($\forall$f:fact) (proposition(f) $\wedge$ tv(denotation(f))$\Leftrightarrow$true)
{For every fact f, f is a proposition and the truth value of the denotation of f is true.}

RULE 1 Any entity may participate in one or more facts.
RULE 2 Any entity may participate in more than one role in one fact.
RULE 3 Each fact in the UoD must be distinct.

TYPE EXAMPLE 1	travel agency has identification number, name, address, telephone number, and fax number
TYPE EXAMPLE 2	season and room type determine rate
TYPE EXAMPLE 3	person has date-of-birth
TYPE EXAMPLE 4	room has room number
INSTANCE EXAMPLE 1	Travel Agency ID 23348, Paula Jones Fun Time Travel, 3421 Holiday St. Tulsa OK 67345, phone 918-555-3668, fax 918-555-4611
INSTANCE EXAMPLE 2	winter single rate is $182
INSTANCE EXAMPLE 3	Steven's date-of-birth is 6th November 1967
INSTANCE EXAMPLE 4	last room on right on third floor has room number 321

NOTE 1 Each role of a entity in a proposition is determined by the semantic predicate of the proposition.

NOTE 2 A fact may be represented by a sentence.

NOTE 3 An attribute is a special kind of fact which relates to only one entity.

NOTE 4 A relationship is a special kind of fact which relates two or more entities.

7.3
constraint
a fact that must always be true

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall c{:}constraint)\ (fact(c) \wedge ([]\ tv(c) \Leftrightarrow true))$
{For every constraint c, c is a fact and it is necessary that the truth value of c is true.}

RULE 1 A constraint must contain at least one entity that is a proposition type, and the constraint must determine the possible truth values of instances of that proposition.
RULE 2 A constraint must define valid UoD states by constraining the possible truth values of propositions.
RULE 3 Every constraint must be distinct.

TYPE EXAMPLE 1	person has only one date-of-birth
TYPE EXAMPLE 2	a reservation for a date cannot be accepted more than 13 months before the date
TYPE EXAMPLE 3	a manager of a project cannot be a worker on the same project
INSTANCE EXAMPLE 1	Steven can only be born on 06 November 1967
INSTANCE EXAMPLE 2	a reservation for September 1999 cannot be accepted in July 1998
INSTANCE EXAMPLE 3	manager Martin manages Project Seascape and cannot work on Project Seascape as a worker

NOTE Specializations of constraint include {unique, mandatory, exclusion, joint exclusion, totality, value, etc.}.

7.4
event
the change from one UoD state to the immediately succeeding UoD state

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall e{:}event)(\ \exists u_1,u_2{:}UoD\ state)\ (e \Leftrightarrow <u_1, u_2> \wedge u_1 \sim \Leftrightarrow u_2 \wedge immsuccessor\ (u_1,u_2))$

{For all events e, there exist UoD states u_1 and u_2 such that e is the sequence of u_1 and u_2 and u_1 is not equal to u_2 and u_2 is the immediate successor of u_1.}

RULE Every event must be distinct.

TYPE EXAMPLE 1 guest enters hotel
TYPE EXAMPLE 2 agent telephones

INSTANCE EXAMPLE 1 Baba arrives at Hotel Yosemite
INSTANCE EXAMPLE 2 Sato rings 234-5000

NOTE The intrinsic nature of the event is not the beginning and ending states, but the change itself.

7.5
process
a chosen set of events

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall p{:}process)\ (\exists e_1..e_n{:}event)\ p \Leftrightarrow \{e_1..e_n\}$
{For all processes p, there exist events e_1 to e_n such that p is the set of events e_1 to e_n.}

RULE Every process must be distinct.

TYPE EXAMPLE 1 check-in guest
TYPE EXAMPLE 2 agree reservation

INSTANCE EXAMPLE 1 check Baba into room 321
INSTANCE EXAMPLE 2 agree reservation for Robert

NOTE 1 A process may use inputs.

NOTE 2 The events may be related by a common concern (e.g., produce an output).

NOTE 3 A process may be triggered by an event.

NOTE 4 In a system there may be many interacting processes.

NOTE 5 A process may be discrete or continuous.

NOTE 6 The events within a process may take place in sequence, in parallel, or in any combination.

7.6
trigger
the role played by an event in starting a process

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall t{:}trigger)\ (\exists p{:}process)\ (\exists e{:}event)((e \Leftrightarrow t) \wedge p \Leftrightarrow \langle e_1..e_n \rangle \wedge immsuccessor\ (t,e_1))$
{For all triggers t, there exists a process p and an event e such that e is a trigger and p is a sequence of events e_1 to e_n and e_1 is an immediate successor of t.}

RULE 1 Every trigger must be distinct.
RULE 2 A process type may have one or more possible trigger types, but a process instance may be triggered by no more than one trigger instance.

RULE 3 A single event may play more than one trigger role, each to a different process.

TYPE EXAMPLE 1 guest enters hotel triggers check-in guest
TYPE EXAMPLE 2 agent telephones triggers agree reservation

INSTANCE EXAMPLE 1 Wolfgang arrives at Hotel Yosemite triggers check Wolfgang into room 321
INSTANCE EXAMPLE 2 Sato rings 234-5000 triggers agree reservation for Robert

NOTE 1 While it may be tempting to want to model a process as the trigger of another process, it will always be possible and usually more precise to model the situation by specifying the particular event in the process as the trigger.

NOTE 2 While it would be possible to model the trigger as starting the first event rather than the whole process, the latter is considered more realistic.

7.7
agent
an entity in a role that has the power to perform, or cause the performance of, one or more processes

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall a: \text{agent})\ (\exists p: \text{process})\ (\text{performs}\ (a,p))$
{For all agents a, there exists a process p such that a performs p.}

RULE 1 A process type may have one or more possible agent types, but a process instance may have as its agent no more than one agent instance.
RULE 2 Every agent must be distinct.

TYPE EXAMPLE 1 hotel receptionist is agent for check-out process
TYPE EXAMPLE 2 hurricane is agent for hotel closing

INSTANCE EXAMPLE 1 Rita is agent for check-out of Bill
INSTANCE EXAMPLE 2 Hurricane Andrew is agent for Miami Beach Hotel closing

7.8
input
an interaction between a entity and a process, where the entity exists before the interaction and is used by the process

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall i: \text{input})\ (\exists p: \text{process})\ (\text{exists-before}\ (i, p) \wedge \text{uses}\ (p, i))$
{For all inputs i, there exists a process p such that i exists before p and p uses i.}

RULE 1 Every input must be distinct.
RULE 2 A process may have one or more input(s).
RULE 3 An entity type may be input to one or more process type(s), but an entity instance may in some cases be restricted to serve as input to only one process instance.

TYPE EXAMPLE 1 name and address is input to a reservation
TYPE EXAMPLE 2 credit card is input to check-in process

INSTANCE EXAMPLE 1 Vanderlei located at Rua Sete de Abril in Curitiba, Brazil is input to reservation at Wawona Hotel
INSTANCE EXAMPLE 2 Diners card 3634 123456 1234 is input to check-in of David

NOTE 1 The entity which is input to a process may or may not be affected by the interaction.

NOTE 2 An input is typically a UoD entity or a fact.

7.9
output
an interaction between a process and a entity such that the entity is either created or its entity state is changed by the process

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall o: \text{output})\ (\exists p: \text{process})$ (created-by (o, p) $\vee$ changed-by (entity state(o), p)
{For all outputs o, there exists a process p such that o is created by p or the entity state of o is changed by p.}

RULE 1 Every output must be distinct.
RULE 2 A process may have one or more output(s).
RULE 3 An entity type may be an output of one or more process type(s), but an entity instance may in many cases be restricted to be the output of only one process instance.

TYPE EXAMPLE 1	confirmation is a possible output of a reservation
TYPE EXAMPLE 2	invoice is output of check-out process
INSTANCE EXAMPLE 1	reservation of Vanderlei is confirmed at Wawona Hotel
INSTANCE EXAMPLE 2	invoice for $999.00 is output of check-out of David

NOTE An output is typically a UoD entity or a fact.

7.10
message
a fact that is an output of one process and an input of zero, one, or more processes

FORMALIZED NATURAL LANGUAGE DEFINITION
$(\forall\ m{:}\text{message})\ (\exists p_1..p_n{:}\text{process})$ (fact(m) $\wedge$ ([]output(m, p_1) $\wedge$ $\Diamond$input(m, $p_2..p_n$)))
{For all messages m, there exist processes p_1 to p_n and m is a fact and it is necessary that m is an output of p_1 and it is possible that m is an input of p_2 to p_n.}

RULE Every message must be distinct.

TYPE EXAMPLE 1	credit card details, output from guaranteed reservation process, input to no-show charge process
TYPE EXAMPLE 2	acknowledgment of payment, output from accounts receivable process, input to debtor (not a process)
INSTANCE EXAMPLE 1	card number 3634 123456 1234, output from reservation for Peter, input to charge to Peter's account
INSTANCE EXAMPLE 2	acknowledgment of $999.00 payment, output from accounts receivable, input to David

8 Conformance

An implementation claiming conformance to this International Standard:

a) shall conform to the semantics of the concepts, principles and architecture specified in 5. and 6.; and

b) shall support the CS modelling constructs defined in 7.

A CSMF implementation may claim conformance to this International Standard if each of the CS modelling constructs is provided, even though one or more may be identified using terms which are different from those used in this International Standard.

A CSMF implementation may alternatively claim conformance to this standard if a mapping can be demonstrated for each of the CS modelling constructs, even though one or more have no direct equivalent in the implementation,

Conformance can be claimed to a subset of the CS modelling constructs in 7. A conforming implementation may provide additional facilities or options not specified in this International Standard provided that these do not conflict with the conformance requirements of this International Standard.

Nothing in this International Standard shall preclude the possibility of an implementor developing a product which claims conformance both to this International Standard and to other relevant database, programming language, or open systems interconnection standards.

Annex A
(informative)

Bibliography

IEEE 1175:1991, *Standard reference model for computing system tool interconnections*

ISO 10303-11:1994, *Industrial automation systems and integration -- product data representation and exchange -- part 11: description methods: the EXPRESS language reference manual*

ISO/IEC 7498-1:1994, *Information technology -- open systems interconnection (OSI) -- basic reference model: the basic model*

ISO/IEC 9075:1992, *Information technology -- database languages -- SQL*

ISO/IEC 10027:1990, *Information technology -- information resource dictionary systems (IRDS) framework*

ISO/IEC 10032:1995, *Information technology -- reference model of data management*

ISO/IEC 10165-1:1993, *Information technology -- open systems interconnection -- management information services -- structure of management information: management information model*

ISO/IEC 10165-7:1996, *Information technology -- open systems interconnection -- structure of management information: general relationship model*

ISO/IEC 10181-1:1996, *Information technology -- open systems interconnection -- security frameworks for open systems: overview*

ISO/IEC 10728:1993, *Information technology -- information resource dictionary system (IRDS) services interface*

ISO/IEC DIS 10746-1, *Information technology -- basic reference model of open distributed processing -- part 1: overview*

ISO/IEC 10746-2:1996, *Information technology -- basic reference model of open distributed processing -- part 2: foundation*

ISO/IEC 13719-1:1995, *Information technology -- portable common tool environment (PCTE) -- part 1: abstract specification*

ISO/IEC JTC1/SC21 Report N236:1985, *Assessment guidelines for conceptual schema language proposals*

Annex B
(informative)

Mathematical conventions and notation

B.1 Document language

In this document, the language used to define the CSMF constructs in 7. is natural language, augmented with a more formalized restatement of the natural language definition using mathematical symbols and variables introduced in 4.2. This annex is an informative elaboration of the notation in 4.2.

B.2 Mathematical symbols and terminology

B.2.1 Sets

A *set* is an arbitrary collection of *elements*, which may be real or imaginary, tangible or abstract. Curly braces are used to enclose a set specification. For small, finite sets, the specification of a set can be an exhaustive list of all its elements: {1, 97, 63, 12}. This specifies a set consisting of the four integers 1, 97, 63, and 12. The order of listing the elements is immaterial, and the following specification is equivalent to the one above: {12, 63, 97, 1}. If the set is very large, like the set of all mammals, a complete listing is impossible. For such sets, the specification must state some *rule* or *property* that determines which elements are in the set: {*x* | Vertebrate(*x*) WarmBlooded(*x*) HasHair(*x*) Lactiferous(*x*)}. In this notation, the variable *x* is called the *bound variable* and the statement following the vertical bar is a *predicate* that is true of all permissible values of the bound variable. This notation specifies the set of all *x* for which the predicate is true, in this case the set of all *x* such that *x* is vertebrate, *x* is warm blooded, *x* has hair, and *x* is lactiferous. A given set may be specified in more than one way. The following four specifications all determine the same set:

- {1, 2, 3}
- {*x* | *x* is an integer and 0<*x*<4}
- {*x* | *x* is a positive integer, *x* divides 6, and $x \neq 6$}
- {*x* | *x*=1 or *x*=2 or *x*=3}

A set specification that lists all elements explicitly is called a definition by *extension* A specification that states a property that must be true of each element is called a definition by *intension* In any application of set theory, there are two privileged sets: the *empty set* {}, which contains no elements at all, and the *universal set* **u** which contains every element that is being considered. Of all the operators that deal with sets, the most basic is $\in$ which states whether a particular element is in a set: the notation $x \in S$ means that *x* is an element of the set *S*; it may also be read *x is a member of the set S* or simply *x is in S* All other operators on sets can be defined in terms of $\in$. Let *A* and *B* be any two sets. Following are the common operators of set theory; listed for each one is its name, standard symbol, informal English definition, and formal definition in terms of $\in$:

- Union $A \wedge B$ is the set that contains all the elements in either *A* or *B* or both: $A \vee B = \{x \mid x \in A \text{ or } x \in B\}$.
- Intersection $A \wedge B$ is the set that contains all the elements that are in both *A* and *B:* $A \wedge B = \{x \mid x \in A \text{ and } x \in B\}$.
- Complement. -*A* is the set that contains everything in the universal set that is not in *A*: $-A = \{x \mid x \in U \text{ and not } x \in A\}$.
- Difference *A*-*B* is the set that contains all the elements that are in *A* but not in *B*: $A\text{-}B = \{x \mid x \in A \text{ and not } x \in B\}$.
- Subset. $A \subset B$ means that every element of *A* is also an element of *B*: if $x \in A$, then $x \in B$. In particular, every set is a subset of itself: $A \subset A$
- Proper subset. *A* is a proper subset of *B* if $A \subset B$ and there is at least one element of *B* that is not in *A*: if $x \in A$, then $x \in B$; and there exists some *B* where $B \in B$, but not $B \in A$.

- Superset. *A* is a superset of *B* if *B* is a subset of *A*.
- Empty set. The empty set has no elements: for every *x*, it is false that $x \in \{\}$. The empty set is a subset of every set, including itself: For every set *A*, $\{\} \subset A$
- Disjoint sets. Two sets *A* and *B* are said to be *disjoint* if their intersection is empty: $A \wedge B = \{\}$.

A set has no duplicate elements. For some applications, duplicate elements may be useful. Therefore, a *bag* is defined as a collection of things with possible duplicates. Since there may be more than one occurrence of a given element *x*, the operator # is used as a generalization of the element operator $\in$: *x#A* is the number of times the element *x* occurs in the bag *A*. A common reason for saving duplicates is to compute averages: if four people have heights of 178cm, 184cm, 178cm, and 181cm, then the set of those numbers is {178, 181, 184} with the average 181; but the bag of the numbers is {178, 178, 181, 184} with average 180.25.

A *sequence* is an ordered bag. To distinguish ordered sequences from unordered sets and bags, the elements of a sequence are enclosed in angle brackets: <178, 184, 178, 181>; the empty sequence is written <>. A sequence of two elements is sometimes called an *ordered pair*; a sequence of three elements, a *triple*; a sequence of four, a *quadruple*; a sequence of five, a *quintuple*; and a sequence of *n* elements, an *n- tuple*.

New sets may be created by combining elements from other sets. The *cross product* or *Cartesian product* of two sets *A* and *B*, written *A*x*B*, is the set of all possible ordered pairs with the first element of each pair taken from *A* and the second element from *B*. If *A* is the set {1,2} and *B* is the set {a,b,c}, then *A*x*B* is the set,

{<1,a>, <1,b>, <1,c>, <2,a>, <2,b>, <2,c>}

With the notation for defining a set by intension, it is possible to give a general definition for the cross product *A*x*B* of any sets *A* and *B*:

$\{\langle x,y\rangle \mid x \in A \text{ and } y \in B\}$

The cross product can also be extended to three or more sets. The product *A*x*B*x*C* is defined as a set of triples: $\{\langle x,y,z\rangle \mid x \in A,\ y \in B, \text{ and } z \in C\}$

B.2.2 Functions

A *function* is a rule for mapping the elements of one set to elements of another set. The notation $f: A \Rightarrow B$ means that *f* is a function that maps any element x in the set *A* into some element $f(x)$ in the set *B*. The set *A* is called the domain of *f*, and *B* is called the range of *f*. The element *x* is called the argument, and *f*(x) is called the *result* or the *image* of *x* under the mapping *f*.

Functions may have more than one argument. A function of two arguments whose first argument comes from a set *A*, second argument from a set *B*, and result from a set *C* is specified $f: A x B \Rightarrow C$. A function with one argument is called *monadic*, with two arguments *dyadic*, with three arguments *triadic*, and with *n* arguments *n- adic.*

When a function $f: A \Rightarrow B$ is defined as a rule for mapping *A* to *B*, it is said to be defined by *intension*. An alternate way of defining a function is to specify a set of ordered pairs, where the first element of each pair is an element *x* of *A*, and the second is the image *f*(*x*) in the set *B*:

$\{\langle a_1,b_1\rangle, \langle a_2,b_2\rangle, \langle a_3,b_3\rangle,...\}$

Such a list of ordered pairs is called the extension of the function *f*.

To distinguish the intension and extension of functions and to formalize the rules for defining them, the system of *lambda calculus*, uses the Greek letter lambda λ to indicate the parameters of a function.

A *lambda expression e* consists of the letter λfollowed by a variable symbol, called the formal parameter of *e* , followed by a statement s, called the body of e.

A function may be defined by an equation that states that some variable symbol equals some lambda expression. Following is the definition of a function named *g*:

g. =λx($2x^2$ + 3*x* - 2).

Given this definition, a term *g* (5) can be expanded by replacing it with the body of the definition and substituting the argument 5 for every occurrence of the formal parameter *x*. In this case, *g*(5) would be replaced by the expression ($2x^5$ + 3x5 - 2), which would evaluate to 63.

B.2.3 Graphs

In diagrams, a graph is normally drawn as a network of nodes connected by arcs. Formally, however, graphs are defined without any reference to a diagram. A *graph G* is defined as a set *N*, whose elements are called *nodes*, and a set *A*, whose elements are called *arcs*. Every arc in *A* is a pair of nodes from the set *N*. A *directed graph* is one in which the arcs are ordered pairs: the arc <*a*,*b*> is considered to be distinct from <*b*,*a*>. An *undirected graph* is one in which the arcs are unordered: <*a*,*b*> and <*b*,*a*> are considered to be alternative ways of specifying the same arc. If e is the arc <*a*,*b*>, the nodes *A* and *B* are said to be *endpoints* of e, and *e* is said to *link a* and *b*. If e is an arc of a directed graph, then the first endpoint *a* is called the source of e, and the second endpoint *b* is called the *target* of e.

A *loop* is an arc e whose endpoints are the same node: e=<*a*,*a*>.

A *path* through a graph is a sequence of nodes <a_0 , ... , a_n > for which any two adjacent nodes a_i and a_{i+1} are linked by some arc. A path that contains *n*+1 nodes must traverse *n* arcs and is therefore said to be of length *n*. A path with only one node <a_0> is of length 0. If the first and last nodes of a path are the same, but all other nodes are distinct, then the path is called a *cycle*. Every loop is a cycle of length 1.

If G is a directed graph, then a path in G may or may not observe the ordering of the arcs. A path in G is said to be *directed* if adjacent nodes occur in the same order in which they occur in some arc of G: if :a_i and a_{i+1} are adjacent nodes on the path, then the ordered pair <a_i , a_{i+1}> must be an arc of G. An arc of a directed graph is like a one-way street, and a directed path obeys all the one- way signs. An undirected path through a directed graph is one that ignores the traffic signs.

A graph is *connected* if there exists at least one path (directed or undirected) between any two nodes. If it is not connected, then it breaks down into disjoint *components*, each of which is connected, but none of which has a path linking it to any other component. A *cutpoint* of a graph is a node, which when removed, causes the graph (or the component in which it is located) to separate into two or more disconnected components.

Certain special cases of graphs are important enough to be given special names: an acyclic graph is one that has no cycles, and a *tree* is an acyclic connected graph for which the path between any two nodes is unique. The most common trees are *rooted trees*, to which the following apply:

- The arcs of a rooted tree are directed.
- If <*a*,*b*> is an arc of the tree, the node *a* is called the *parent* of *b*, and *b* is a *child* of *a*..
- There is a privileged node called the root, which has no parent.
- Every node except the root has exactly one parent.
- A node that has no child is called a *leaf* .

A *binary tree* is a rooted tree where every node that is not a leaf has exactly two children. In a binary tree, the two children of each node are usually designated as the *left child* and the *right child*. A *forest* is a collection of disconnected trees; a *chain* is a tree with no branches - all the nodes lie along a single path; and a *seed* has only one node and no arcs.

B.2.4 Relations

A *relation* is a function of one or more arguments whose range is the set of *truth values*, {true,false}. An example of a dyadic or binary relation is the function less than represented by the operator symbol <. Its domain is the set of integers $\mathbb{Z}$:

<: $\mathbb{Z}$x$\mathbb{Z}\rightarrow${true,false}

The < relation is not symmetric: 5<12 gives the result true, but 12<5 gives the result false. Relations are often written as infix operators with special symbols, x<*y* or x$\in$ S; they may sometimes be represented by single letters, R(*x*,*y*) or S(*x*,*y*,*z*); or they may be represented by arbitrarily long alphanumeric strings, mother(*x*,*y*) or between(*x*,*y*,*z*). The term *predicate* is a synonym for relation. A predicate with only one argument is also called a property.

As with other functions, relations may be defined either by intension or by extension. An intensional definition is a rule for computing a value true. or false. for each possible input. An extensional definition is a set of all combinations of arguments for which the relation is true; for all other combinations, the relation is false. In a database, a *stored relation* is one whose values are listed by extension, and a *virtual relation* l is computed by some rule. In theory, implementation issues are irrelevant. In practice, the question of how a relation is implemented is of vital importance: the relation *x*<*y* is easy to compute, but it would require an infinite amount of space to store.

Graphs and binary relations are equivalent ways of describing the same mathematical structures. Historically, graphs are more closely associated with geometric properties that can be seen from diagrams, and relations are associated with more abstract mathematics and logic. But every binary relation can be represented as a graph, and every graph defines a binary relation Let G be a graph, and let the symbol $\mathfrak{R}$ represent the equivalent binary relation. If *x* and *y* are nodes in the graph G, define $x\mathfrak{R}y$=true if the pair <*x*,*y*> is an arc of G, and $x\mathfrak{R}y$=false if <*x*,*y*> is not an arc of G. If the graph is undirected, then $\mathfrak{R}$ is symmetric because it obeys the constraint $x\mathfrak{R}y=y\mathfrak{R}x$. Following are some common properties of relations, their associated constraints, and some examples defined over the set $\mathbb{P}$of people:

Name	Constraint	Example
Reflexive	$\forall x$, $x\mathfrak{R}x$	*x* is as old as *y* l
Irreflexive	$\forall x$, ~($x\mathfrak{R}x$)	*x* is the mother of *y*
Symmetric	$x\mathfrak{R}y \Rightarrow y\mathfrak{R}x$	*x* is the spouse of *y*
Asymmetric	$x\mathfrak{R}y \Rightarrow$ ~($y\mathfrak{R}x$)	*x* is the husband of *y*
Antisymmetric	$x\mathfrak{R}y$ & $y\mathfrak{R}x \Rightarrow x=y$	*x* was present at *y*'s birth
Transitive	$x\mathfrak{R}y$ & $y\mathfrak{R}z \Rightarrow x\mathfrak{R}z$	*x* is an ancestor of *y*

Certain combinations of these properties are also common. A partial ordering, represented by the symbol $\leq$ is a binary relation that is reflexive, antisymmetric, and transitive. The subset relation $\subset$ is the most common partial ordering over sets. It is antisymmetric because $x\subset y$ and $y\supset x$ imply that *x*=*y* . A linear ordering is a partial ordering where x$\leq$*y* or *y*$\leq$x for every pair x and *y*. For real numbers, $\leq$ represents the linear ordering *less than or equal to*. The subset relation is only a partial ordering because there are many sets for which neither $x\subset y$ nor $y\supset x$ is true.

A *mathematical structure* is a set together with one or more relations and operators defined over the set. A *lattice* is a structure consisting of a set *L*, a partial ordering $\leq$, and two binary operators $\wedge$ and $\wedge$. If *a* and *b* are elements of *L*, $a\wedge b$ is called the *greatest lower bound* or *infimum* of *a* and *b*; $a\vee b$ is called the *least upper bound* or *supremum* of *a* and *b*. These operators satisfy the following axioms:

- For any *a* and *b* in *L*, $a\wedge b\leq a$, $a\wedge b\leq b$, and if *c* is any element of *L* for which $c\leq a$ and $c\leq b$, then $c\leq .a\wedge b$
- For any *a* and *b* in *L*, $a\leq a\wedge b$, $b\leq a\wedge b$, and if *c* is any element of *L* for which *a*$\leq$c and $b\leq c$, then $a\vee b\leq c$.

The symbols $\cap$ and$\cup$are the same as the symbols for intersection and union of sets. This similarity is not an accident because the set of all subsets of a universal set U forms a lattice with the subset relation $\subset$ as the partial ordering. A *bounded lattice* is one with a top T and a bottom $\perp$ where for any element *a* in the lattice, $\perp\leq a\leq$T. All

finite lattices are bounded, and so are many infinite ones. In a lattice of subsets, the universal set U is T and the empty set {} is ⊥.
If a binary relation is reflexive, symmetric, and transitive, it is called an *equivalence relation.* The archetype of an equivalence relation is equality: it is reflexive, because *x*=*x*; it is symmetric, because *x*=*y* implies *y*=*x*; and it is transitive, because *x*=*y* and *y*=*z* imply *x*=*z* . But there are many other equivalence relations. For example, *born under the same sign of the zodiac* is an equivalence relation defined over the set P of people.

Whenever an equivalence relation is defined over a set *S*, it divides the set *S* into subsets called *equivalence classes*. The zodiac relation, for example, divides the set P of people into 12 equivalence classes that have the traditional labels Aries, Taurus, ..., Pisces. In general, if $\Re$ is any equivalence relation defined over a set *S*, then two elements *x* and *y* are in the same equivalence class if and only if $x \Re y$:

B.2.5 Symbolic logic

Symbolic logic has two main branches: *propositional logic* and *predicate logic*. Propositional logic deals with statements or *propositions* and the connections between them. The symbol *m*, for example, could represent the proposition, *Lillian is the mother of Leslie*. Predicate logic, however, would represent that proposition by a predicate mother(*x*,*y*) applied to the two individuals: mother(Lillian,Leslie). Whereas propositional logic represents a complete statement by a single symbol, predicate logic analyzes the statement into finer components. Besides symbols for propositions, propositional logic also includes symbols for operators like *and, or, not*, and *if- then*. Let *p* be the proposition *The sun is shining*, and let *q* be the proposition *It is raining*. The most commonly used operators in the propositional logic correspond to the English words *and, or, not*, *if- then*, and *if- and- only- if*:

- Conjunction (and) — $p \wedge q$ represents the proposition *The sun is shining, and it is raining.*
- Disjunction (or) — $p \vee q$ represents *The sun is shining, or it is raining*.
- Negation (not) — $\sim p$ represents *The sun is not shining*.
- Material implication (if- then) — $p \Rightarrow q$ represents *If the sun is shining, then it is raining.*
- Biconditional (if- and- only- if) — $p \Leftrightarrow q$ represents *The sun is shining if and only if it is raining*.

The operators ∧, ∨, ~, ⇒, and ⇔ are called *Boolean operators*.

If certain Boolean operators are taken as primitives, the others can be defined in terms of them. One common choice of primitives is the pair ~ and ∧, Then the other operators are defined as follows:

- $p \vee q$ is equivalent to $\sim(\sim p \wedge \sim q)$
- $p \Rightarrow q$ is equivalent to $\sim(p \wedge \sim q)$
- $p \Leftrightarrow q$ is equivalent to $\sim(p \wedge \sim q) \wedge \sim(\sim p \wedge q)$

In propositional logic, the proposition *All peaches are fuzzy* may be represented by a single symbol *p*. In predicate logic, however, the internal structure of the proposition is represented in detail. Then *p* would be represented by a more detailed *formula*:

$$\forall x(\text{peach}(x) \Rightarrow \text{fuzzy}(x)).$$

The symbol $\forall$ is called the *universal quantifier*, and the symbols peach and fuzzy are predicates like the ones described in Section 1.5. The combination $\forall x$ is commonly read *for all x*, the combination peach(*x*) is read *x is a peach*, and the combination fuzzy(x) is read *x is fuzzy* . The entire formula, therefore, may be read *For all x, if x is a peach, then x is fuzzy* . Since predicates (or relations) are functions that yield truth values as results and since the Boolean operators are functions that take truth values as inputs, predicates can be combined with the same operators used in propositional logic.

Predicate logic has one additional symbol, the existential quantifier represented as ∃; the combination ∃*x* may be read *there exists an x such that*. The following formula uses an existential quantifier:

~∃*x*(peach(*x*) ∧ ~fuzzy(*x*)).

This may be read *It is false that there exists an x such that x is a peach and x is not fuzzy* . Formulas with more than one quantifier are possible. The English statement *For any integer x, there is a prime number greater than x* is represented as,

∀*x*∃*y*(integer(*x*) ⇒ (prime(*y*) ∧ *x*<*y*)).

Literally, this formula may be read *For all x, there exists a y such that if x is an integer, then y is prime and x is less than y* .

The two types of quantifiers, Boolean operators, variables, predicates, and the rules for putting them together in formulas make up the entire notation of *first- order predicate logic*, which is also known as *first- order logic* . It is called *first order* because the range of quantifiers is restricted to simple, unanalyzable individuals. Higher- order logic also allows function symbols and predicate symbols to be governed by the quantifiers. An example of a higher-order formula is the *axiom of induction*:

∀P((P(0) ∧ ∀*n*(P(*n*) ⇒ P(*n*+1)) ⇒ ∀*n*P(*n*)).

This formula may be read, *For all predicates P, if P is true for 0, and for all n, P(n) implies P(n+1), then P is true for all n*. This is the only axiom for arithmetic that requires more expressive power than first- order logic.

Any of the functions, operators, relations, and predicates defined above can also be used in the formulas of first-order logic. Following are the *formation rules* that define the syntax of formulas:

- A *term* is either a *constant* like 2, a *variable* like x, or an *n*- adic function symbol applied to *n* arguments, each of which is itself a term.
- An *atom* is either a single letter like *p* that represents a proposition or an *n*- adic predicate symbol applied to *n* arguments, each of which is a term.
- A *formula* is either an atom, a formula preceded by ~, any two formulas *A* and *B* together with any dyadic Boolean operator ℜ in the combination (*A* ℜ*B*), or any formula *A* and any variable *x* in either of the combinations ∃*xA* or ∀*xA* .

To be explicit, every new variable, constant, or function that is introduced will be preceded by a declaration, such as "Let x be a variable". or "Let *C* be a constant".

B.2.6 Grammars

A grammar has two kinds of symbols: *terminal symbols* like *the, dog,* or *jump*, which appear in the sentences of the language that is being defined; and *nonterminal* symbols like N, NP, and S, which represent grammatical categories, such as noun, noun phrase, and sentence. The grammar rules, which are called *production rules*, state how the nonterminal symbols are transformed in order to generate sentences of the language. Terminal symbols are called *terminal* because no production rules apply to them: when a derivation generates a string consisting only of terminal symbols, it must terminate. Nonterminal symbols, however, keep getting replaced during a derivation. A *context- free grammar* G has four components:

- A set of symbols *T*, called the *terminal symbols*,
- A set of symbols *N*, called the *nonterminal symbols*, with the restriction that *T* and *N* are disjoint: *T*∧*N*={}.
- A special nonterminal symbol *S*, called the *start symbol*,
- A set of *production rules P*, of the form: *A* ⇒*B* , where *A* is a nonterminal symbol and *B* is a string of one or more terminal or nonterminal symbols.

The start symbol corresponds to the highest level category that is recognized by the grammar, such as *sentence*. The production rules generate sentences by starting with the start symbol and systematically replacing nonterminal symbols until a string consisting only of terminals is derived.

Some convention must be adopted for distinguishing terminals from nonterminals. To be explicit, this document will enclose terminal symbols in double quotes: ".the" or "|". To illustrate the formalism, the following grammar defines a small fragment of English:

- Terminal symbols *T*: {"the", "a", "cat", "dog", "saw", "chased"}
- Nonterminal symbols *N*: {S, NP, VP, DET, N, V}
- Start symbol *S*: S

The set T. defines a 6- word vocabulary, and the set *N* defines the basic grammatical categories. The starting symbol S represents a complete sentence. The symbol NP represents a noun phrase, VP a verb phrase, DET a determiner, N a noun, and V a verb. The following 9 production rules determine the grammatical combinations for this language:

S $\Rightarrow$ NP VP	N $\Rightarrow$"cat"
NP $\Rightarrow$ DET N	N $\Rightarrow$ "dog"
VP $\Rightarrow$ V NP	V $\Rightarrow$ "saw"
DET $\Rightarrow$ "the"	V $\Rightarrow$ "chased"
DET $\Rightarrow$ "a"	

This grammar can be used either to generate or to parse sentences like *A dog saw a cat. The cat saw the dog. The dog chased the cat. The cat chased the dog.*